SZAKDOLGOZAT

Varga Marcell

2014

Pannon Egyetem

Matematika Tanszék

Mérnök informatikus BSc szak

SZAKDOLGOZAT

Képfeldolgozást támogató keretrendszer és modulok készítése

Varga Marcell

Témavezető: Lipovits Ágnes

2014

Ide jön az eredeti vagy a fénymásolt feladatkiírás.

Nyilatkozat

Alulírott Varga Marcell diplomázó hallgató kijelentem, hogy a szakdolgozatot a Pannon Egyetem Matematika Tanszékén készítettem Mérnök informatikus BSc szak (BSc in Computer Engineering) megszerzése érdekében.

Kijelentem, hogy a szakdolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a szakdolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2014. május 02.

Aláírás

Alulírott Lipovits Ágnes témavezető kijelentem, hogy a szakdolgozatot Varga Marcell a Pannon Egyetem Matematika Tanszékén készítette Mérnök informatikus BSc szak (BSc in Computer Engineering) megszerzése érdekében.

Kijelentem, hogy a szakdolgozat védésre bocsátását engedélyezem.

Veszprém, 2014. május 02.

Aláírás

Köszönetnyilvánítás

Köszönöm a családomnak a sok türelmet és segítséget, amit kaptam, nélkülük ez a szakdolgozat nem készült volna el.

Köszönöm témavezetőmnek, Lipovits Ágnesnek az elmúlt egy év során adott irány-mutatását.

Végül, de nem utolsó sorban, szeretném megköszönni a szaktársaimnak a bíztatást.

TARTALMI ÖSSZEFOGLALÓ

E szakdolgozat témája egy olyan képfeldolgozást segítő szoftver, amelynek segítségével könnyen egyszerű vizuális eszközökkel többlépcsős feldolgozási műveleteket végezhetünk, akár nagy mennyiségű bemeneti adaton is.

A fejlesztést C++ban Qt-val és OpenCV segítségével végeztem. Ezzel biztosítva azt, hogy az alkalmazás a dinamikusan betölthető moduljai tartalmazzák a képfeldolgozást végző logikai blokkokat.

Kulcsszavak: *szoftverarchitektúra, képfeldolgozás, adatszerkezetek, adatkezelés, Qt, c++, OpenCV*

ABSTRACT

The topic of this thesis is to ...

Keywords: *software-architecture, image processing, data structure, data processing, Qt, c++, OpenCV*

Tartalomjegyzék

1. A feladat összefoglalása

Témám egy olyan képfeldolgozást támogató keretrendszer tervezése és fejlesztése, amely alkalmas képek egyedi vizsgálatára és kötegelt feldolgozására. A feldolgozást végző algoritmusok a dinamikusan betölthető modulokban foglalnak helyet. A rendszer fő haszonélvezője a Pannon Egyetem Képfeldolgozás Kutatólaboratóriuma lesz, de célom, hogy kellően általános rendszer jöjjön létre, amelyet bárki könnyen és egyszerűen használhat, illetve bővítheti saját modulokkal.

1.1. Első lépések

1.1.1. Briefing

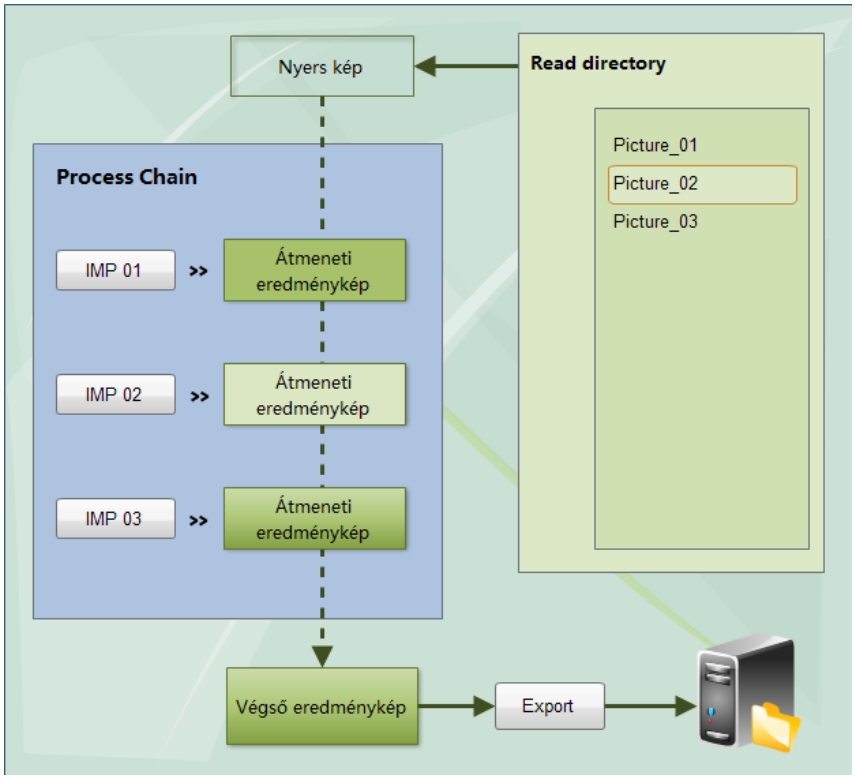
A legtöbb munka során előnyös, ha projekt lényegét megragadva röviden összefoglaljuk a legfontosabb lefutási eset sikeres teljesülését. Erre jó eszköz a rövid (brief[1]) formátumú usecase.
"A felhasználó összeállítja a bemeneti képek, adatforrások listáját. Ezek után meghatározza a feldolgozás lépéseit. Végül megjelöli a kimeneti formát, majd elindítja a feldolgozást. A program egyesével létrehozza a nyers képeket[1]*, majd az adott nyers képen végrehajtja sorrendhelyesen a kijelölt feladatokat, végül a választott kimeneti formába menti ki az eredményképeket*[2]*."*

1.1.2. Egy rövid példa

Az 1. ábrán láthatjuk a rendszer vázlatos működését. Jelen példában egy könyvtárban található összes digitális képet kívánjuk feldolgozni. A beolvasás után a nyers képet átadjuk a feldolgozást végző logikának (Process Chain), ahol jelen példában 3 darab elemi művelet történik (IMP01-03). A feldolgozás utolsó lépése után, a végső eredménykép (jelen esetben) exportálásra kerül, egy a bementi könyvtárral nem megegyező könyvtárba.

[1] Nyersképnek nevezünk minden olyan bemeneti képet, amelyen nem hajtottunk végre semmiféle változtatást.

[2] Átmeneti eredmény képnek nevezzük minden olyan képet, amelyen már végrehajtottunk végbe ment feldolgozás, de nem még nem az összes. Eredmény képnek pedig az olyan képeket nevezzük, amelyeken már lezajlott a feldolgozás



1. ábra: A rendszer vázlatos működése

Fontos megjegyezni, hogy a feldolgozás pipelining [2] jelleget követ, tehát apró elemi lépések sorozata, amelyek kötött sorrendben végzik feladataikat. A műveletekből egy irányított gráf írható fel. Ahol a csúcspontok a műveleti egységek, az irányított élek pedig az adatok áramlása. (Ilyen műveleti módra jó példa különböző grafikus engineknél a fények számítása pl.: [3].)

2. Hasonló célú rendszerek

Következő lépésként megvizsgáltam, hogy milyen hasonló célú szoftverek, illetve szoftvercsomagok találhatóak meg a piacon. Erre azért volt szükség, hogy pontosabb képet kapjak a jelenleg fellelhető megoldásokról, és munkám során az így szerzett pozitív és negatív tapasztalatokat eredményesen hasznosíthassam.

Különböző összehasonlítási szempontokat állítottam fel, melyek lentebb olvashatóak. Az vizsgálat során a személyes benyomáson túl, egyéni véleményeket is figyelembe vettem (pl.: kiadó cégnek vagy alapítványak az ajánlása, vagy független publikáció, újságcikk).

2.1. Összehasonlítási szempontok

2.1.1. Általános tulajdonságok

- Platform: Milyen környezetben és operációs rendszeren használható? Milyen eszközökkel fejlesztették?
  Hordozhatósági szempontonból került be a listára.
- Licence: Milyen licenc alatt került publikálásra?
  Elsősorban pénzügyi és kód újrahasznosíthatósági jellemzők miatt érdekes.
- Cél csoport: A szoftver kinek az igényeinek a kielégítésére törekszik?
  Legtöbb esetben a célcsoport már alapvetően meghatározza, hogy a szoftverbe milyen funkcionalitásokat építünk be, illetve, hogy ezekhez milyen interfészt biztosítunk a jövőbeni felhasználóink részére.
- Támogató: Van hivatalos támogatása? (cég, alapítvány)
  Az esetek jelentős részében megfigyelhető, hogy egy szoftver, szoftvercsomag akkor válik igazán jól támogatottá, ha fejlesztői közösségen kívül egy nagyobb szervezet is gondozásába veszi.
- Felhasználói közösség: Fórum, levelező listák?
  Bármilyen előre nem látható hiba történhet: Ami elromolhat az el is romlik! A fenti csatornákon segítséget kérve nagy eséllyel kaphatunk választ kérdésünkre, és megoldást problémáinkra.
- Plugin rendszer: Plugin betöltésre van lehetőségünk? Saját plugin?
  A képfeldolgozás egy eléggé sokrétű szerteágazó lehetőségeket, funkcionalitásokat magában foglaló szakterület. Így az csak utópisztikus álom, hogy egyszer valaki implementálja az összes funkcionalitást és onnantól kezdve mindenki boldogan használja azokat az idő végezetéig... Ezért ha a program dinamikusan bővíthető (akár a felhasználó által készített bővítményekkel), jelentős előnyt jelent a többi monolitikus rendszerrel szemben.
- Kötegelt feldolgozási lehetőség: Feldolgozhatunk egyszerre nagy mennyiségű képet?
- Automatizálási lehetőségek: Automatizálhatjuk a feldolgozást?
  Ha lehetőségünk van a meglévő egyszerű feldolgozási lépéseket testreszabni,

esetleg összekombinálni akkor az szintén egy jelentős előny lehet, más ilyen lehetőségekkel nem rendelkező szoftverekkel szemben.

- Fejlesztői eszközök: Rendelkezik hivatalos fejlesztői eszközökkel?
- Támogatott bemeneti formátumok köre
- Megjelenítési, vizualizációs lehetőségek listája, módjai
  Hasznos ha több féle interfészt biztosítunk az adott információ megjelenítéshez: más logikai kontextusban helyezve új megfigyeléseket is tehet a felhasználónk.

2.1.2. Képfeldolgozási képességek

- Képjavító eljárások, pl.: élesítés, kontrasztkiegyenlítés
- Geometriai műveletek, pl.: átméretezés, forgatás, tükrözés
- Analizálás, pl.: eltérések detektálása, alacsony szintű képleírók
- Szerkesztési műveletek, pl.: logikai, szöveg, alakzatok elhelyezése
- Színterek közötti konverzió, pl.: RGB $\rightarrow$ HSL, csatornák külön kezelése stb

2.2. Választott szoftverek

- *ImageJ*[4]
  Képfeldolgozást és analizálást végző rendszer, amely a National Institutes of Health fejlesztése. A program első indulásakor látható, hogy itt egy professzionális orvostechnológiai eszközről van szó. Támogatottsága jelentős mind közösségi, mind bővíthetőségi szempontból. Eszközkészletének palettája széleskörű, külön kiemelném Z és T funkciókat.[5]
  A Z funkciók segítségével pl.: MRI-vel készített sorozatos metszeti képeket kezelhetünk könnyedén, lehetőségeinket tovább növeli, hogy a térbeli szervezés mellett még időbeli struktúra felépítésére és kezelésére is lehetőséget ad a program (T funkciók).

- *ImBatch*[6]
  Képfeldolgozást végző rendszer. Célcsoportja egyértelműen egy félprofesszionális felhasználói szint. Tehát itt eleve nem is várunk professzionális analiti-

kai funkciókat. Cserébe kapunk egy szép, letisztult, egyszerű grafikus felhasználói felületet, és egy pár használatot segítő kényelmi funkciót: pl.: Windows helyérzékeny menü intergrációt.

- *OriginLab - Image Processing*[7]
  Az OriginLab szoftver csomag része, amely első sorban tudományos és ipari célközöséget szolgál ki. [8] A korábban tárgyalt rendszerekkel ellentétben ez a szoftver fizetős (21 napos teszt verzió igényelhető). Ára hozza az iparban szokásos szoftver árakat [9], amely személyes felhasználásra kissé borsos, azonban funkcionalitása kárpótolja a felhasználót. Rentgeteg elemzési lehetőség mellett még OriginC-ben saját algoritmusainkat is megvalósíthatunk, a LabView támogatás már majdnem, hogy csak hab a tortán.

2.3. Összefoglalás

2.3.1. Tapasztalatok

A részletes összehasonlítás az 1.1. táblázatból olvasható ki a 7. oldalon.

Az adatsorok elemzése közben, több fontos követelmény, fejlesztési irányvonal körvonalazódott:

- Hordozható legyen több platformra. Hiszen egy laboratóriumban több féle architektúra elő fordul. (Ideális esetben tehát a szoftverünk legyen crossplatform.)
- Nyitott legyen a további fejlesztésekre. A felhasználónak adjuk meg a lehetőséget, hogy testre szabhassa a szoftverünket, vagy akár önmaga is fejlesztővé válhasson, így bővíthesse a pluginek körét, vagy javíthassa a fő programot.
- Szerepeljenek automatizálási lehetőségek. Írhassunk makrókat, vagy vizuális módon szerkeszthessünk algoritmusokat.
- A rendszert ajánlott az adott részterületen leggyakrabban előforduló, és legnépszerűbb ki- és bemeneti adatformátumokra felkészíteni.

- Már gyárilag nagy mennyiségű képjavító, feldolgozó, szerkesztő, elemző és analizáló funkcionalitással érkezzen a szoftver.

2.3.2. Célok

Összegezve láthatjuk, hogy eléggé könnyen lehet már előkészített rendszereket választani a szoftverpiac palettájáról. Ilyenkor jogosan felmerül a kérdés, hogy ez a projekt miben ad többet, mint a jelenlegi lehetőségek?

- Célom, hogy a programba új funkciók integrálása ne ütközzön problémákba, és az ilyen módon implementált funkcióknak a főprogramtól függetlenül is terjesztőknek kell lenniük. Ez természetesen maga után vonzza, hogy a plugin és főprogram közötti interfésznek kellően kompaktnak és univerzálisnak kell lennie, hogy több verzióváltás alatt is változatlan lehessen.

- A sok kis méretű funkció egy idő után kezelhetetlenné válik, ezért legyen lehetőség kategorizálásra.

- Legyen lehetőség átlátható szerkezetű grafikus felhasználói felület használatára. Erre kifejezetten alkalmas a blokkos kapcsolat alapú vizuális szerkesztő. Ez a módszer több különböző technológiai területen sikeresen vizsgázott pl.: Labview blockdiagramjai [11], vagy UDK4 BluePrint Editorja[12] (amely az UDK3 Kismetjének egy tovább fejlesztett változata).

2.4. Hasonló célú rendszerek összehasonlítása táblázat

	ImageJ	ImBatch	OriginLab
Platform	Multi (Java)	Win (C#)	Win (C/C++)
Licence	Public Domain		Fizetős Jogvédett
Cél csoport	professzionális (orvosi)	félprofesszionális (általános)	professzionális (tudományos, ipari)
Támogató	National Institutes of Health	High Motion Software	OriginLab
Felhasználói közösség	wiki, leírások, fejlesztői dokumtáció, levlista, fórum	gyik, leírások, oktató videók	gyik, wiki, leírások, fejlesztői dokumentáció, fórum
Plugin rendszer	igen	igen	igen
Kötegelt feldolgozás	igen (Z-T funkciók)	igen (akár helyérzékeny menü)	igen
Automatizálás	makrók	részben	originC
Fejlesztői eszközök	igen	igen	igen
Bemeneti formátumok	széleskörű (orvosi irány)	széleskörű (általános irány)	széleskörű (ipari irány)
Megjelenítés, GUI	komplex	egyszerű letisztult	komplex
Képjavító eljárások	igen	igen	igen
Geometriai műveletek	igen	igen	igen
Analizálás	igen	nem	igen
Szerkesztési műveletek	igen	igen	igen
Színterek közötti konverzió	igen	igen	igen

1.1. táblázat: Hasonló célú rendszerek összehasonlítása

3. Rendszertervek

A következőkben röviden összefoglalom a szakdolgozat tárgyát képező szoftver terveinek elő- és elkészítésének menetét. A korábbi fejezet végén vázolni kezdtem a programmal szembeni elvárásokat, javasalatokat, ez gyakorlatilag a követelményrendszer felírásának a kezdeti lépése volt. Természetesen a jelenlegi követelményrendszer kialakítását még megelőzte több beszélgetés, konzultáció is.
A megbeszélések során több logikailag összetartozó objektum, folyamat került felírásra. Ezek rendre nevet kaptak a kommunikáció hatékonyságának növelése érdekében. A legfontosabb elenevezéseket a 1.2 táblázat prezentálja.

3.1. Szószedet

Elnevezés	Deffiníció
Bimg	A főprogramnak az elnevezése (képzése a Batch Image szavakból történt szóösszerántással).
Modul	BIMG-n belüli logikai egység.
Plugin	BIMG dinamikus kiterjesztése, az IMP Nodek lelőhelye
IMP vagy Node	Image Process Node, képfeldolgozási alapegységnek tekínthetjük, egy IMP általában egy jól körülhatárol művelet elvégzésére alkalmas. Három típusa van: adat (DataNode), indikátor (IndicatorNode), és feldolgozó (ProcessNode)
Slot	IMP Node be- és kimeneti interfészei. Egy slot egy paramétert kezel és tárol.
Parameter	Tetszőleges típusú adat. (pl.: szám, szöveg, kép)
Slot Connection	Kettő darab azonos típusú paraméter tartalmazó slot között kapcsolat hozható létre. Ezt a kapcsolatot nevezzük Node Slot Connectionnak, vagy IMP Connection-nak. Kapcsolat csak kimeneti és bemeneti slotok között jöhet létre. Az irány kötelezően: Ki-Be.
Process Chain	Node-kat tartalmaz, és vezérli a feldolgozást.

1.2. táblázat: Szószedet

A továbbiakban ezeket az elnevezéseket használom az adott objektumokra.

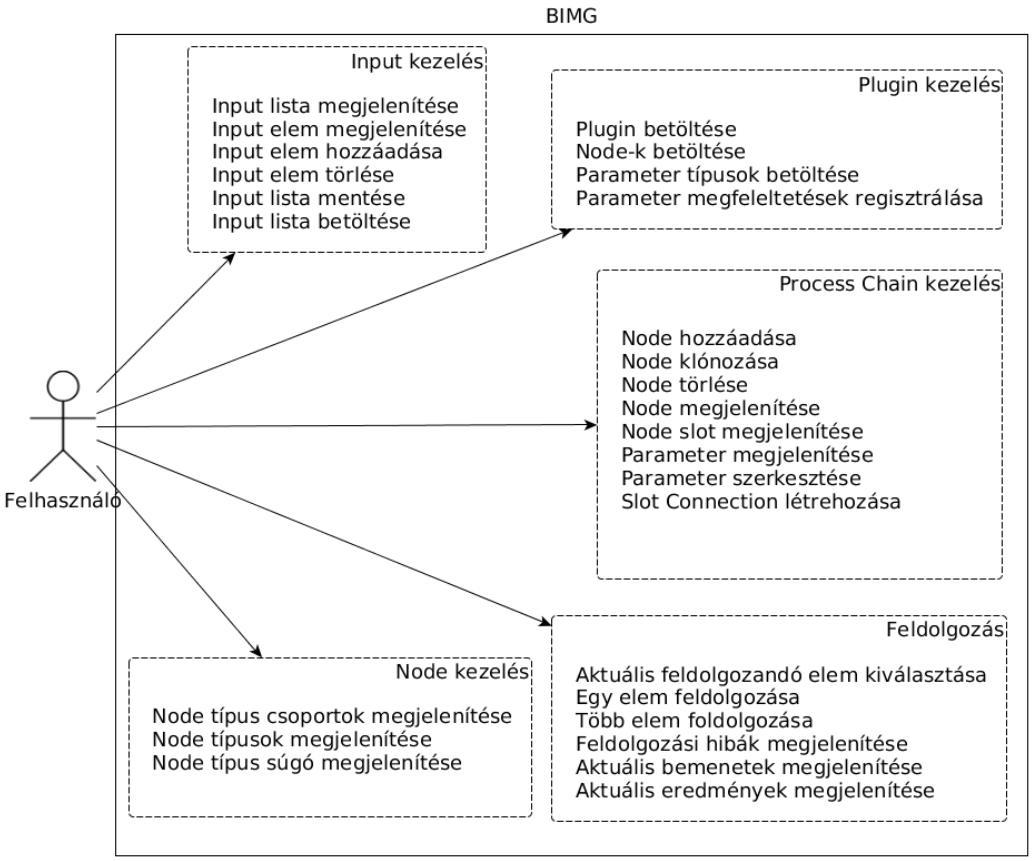
3.2. Követelmény analízis

A követelmény analízist a RUP (Rational Unified Process) metodika FURPS+ rendszerének útmutatása alapján végeztem.
Célom az volt hogy a legtöbb tulajdonság és jellemző feltüntetésre kerüljön amely

szükséges, hogy a szoftver megoldja az adott problémát. [20] Ezek a követelmények jellemzőjüket tekintve lehetnek funkcionális követelmények, és nem funkcionális követelmények.

3.2.1. Funkcionális követelmények

Funkcionális követelményeknek tekintünk minden olyan követelményt, amely a fő termékünkbe valamilyen képességet biztosít. [21] Azaz ide tartozik minden felhasználói, cél feladat és aktivitás. Erre egy kiválló eszköz a usecase, melyet legegyszerűbben a usecasediagramm segítségével tekinthetünk át.
A 2. ábrán megfigyelhető a BIMG sematikus usecase diagramja[3].



2. ábra: A BIMG sematikus usecase diagramja

Az ábráról is egyértelműen leolvashatóak rendszert képző legfontosabb logikai egységek:

[3]Megjegyzés: a vázlatos ábrázolás és a könnyebb szemléltetés miatt a felhasználó csak a funkcionalitások csoportjaival és nem a külön-külön a funkcionalitásokkal került összekötésre.

- Input kezelés: a funkcionalitásokból láthatjuk, hogy egy bemeneti elemekből azaz nyers képekből álló struktúrát tudunk managelni.
- Plugin kezelés: itt töldőnek be a feldolgozó egységek és az ezekhez tartozó esetleges, speciális adattagok is. Fontos megjegyezni, hogy szükség van az adattagok közötti konverziós lehetőségek deffiniálására is, hiszen így lesznek képesek a különböző fejlesztésű node-k egymással hatékonyan kommunikálni és együttműködni.
- Node kezelés: a pluginekből beolvasott node-k kezelése törénik ezen a szinten.
- Process Chain kezelés: Ezen a szinten találjuk meg azokat a funkciókat, amelyek a felhasználó számára lehetőséget adnak a feldolgozási folyamat deffiniálásra, szerkesztésére és tesztelésére.
- Feldolgozás: Ezen a szinten történik a feldolgozás összehangolása és az eredmények vizualizálása.

A teljes diagramm a 19. ábrán tekinthető meg a. A kifejtett usecase a mellékletben csatolva olvasható: *BIMG-usecase.pdf*

3.2.2. Nem funkcionális követelmények

A nem funkcionális követelmények halmazába tartozik minden, olyan követelmény, amely valamilyen korlátot, megszorítást vagy minőség irányú feltételt deffiniál. [22] Természetesen ide a futás idejű követelményeken túl (mint pl.: megbízhatóság, teljesítmény, hibakezelés), ideértjük a fejlesztési követelményeket is (pl.: modularitás, bővíthetőség, kód újrafelhasználás stb). Először összegyűjtöttem a legfontosabb nem funkcionális követelményeket, majd ezeket osztályoztam, hogy az (F)URPS+ metodika melyik osztályába tartoznak. Ehhez jó kiindulási pont volt a témakiírás és a konzultációs beszélgetések.

3.3. A feladat modellezése - Domain modell

A követelmény analízist a megoldandó feladat részletes elemzése és modellezése követte. Erre az egyik legszélesebb körben használt eszköz a domain modell. Elő-

Típus	Követelmény
Modularitás és bővíthetőség	A paraméterezhető képfeldolgozási algoritmusokat dinamikusan betölthető modulok biztosítsák a rendszer számára.
Használhatóság	A könnyen kezelhető grafikus felhasználói felületen legyen mód többlépcsős feldolgozásra.
Támogatás	Amennyiben a felszhasználói felület használata nem teljesen triviális, készüljön hozzá kezelési leírás.
Teljesítmény és megbízhatóság	A rendszer legyen képes nagy mennyiségű bemenet feldolgozására.
Megbízhatóság	Egy esetleges hibás működésű modul ne okozzon rendszer szintű problémát.
Implementáció	A fejlesztés lehetőleg modern, ingyenes, nyíltforrású technológiákkal történjen.
Interfész	Több ki és bemeneti formátum támogatása.

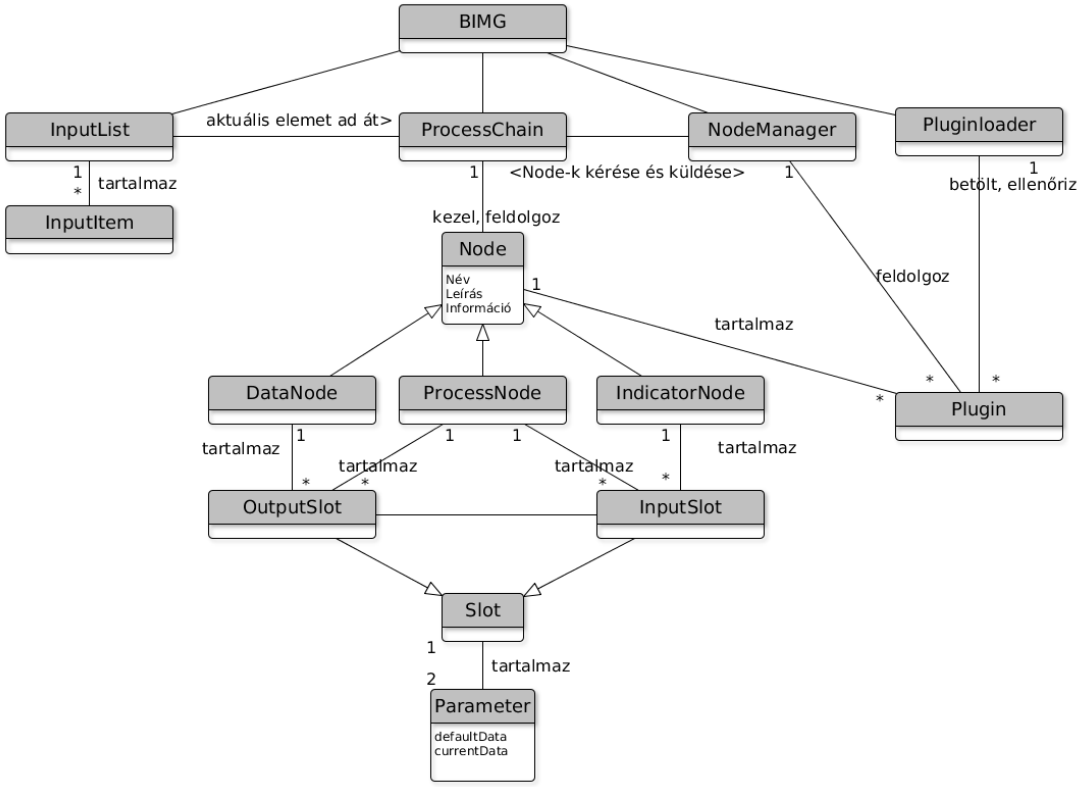
1.3. táblázat: A legfontosabb nem funkcionális követelmények

nye, hogy kevés objektummal egyszerűen, és érthetően tudjuk ábrázolni a megoldani kívánt problémakört és feladatokat.[1]

A problémát bemutató domain modell a 3. ábrán látható. Itt szeretném megjegyezni, hogy ez a modell nem az egész rendszer ábrázolja. Itt csak alaprendszerek vázlatos bemutatására szorítkozom: Node-Slot-Parameter, Process Chain, Input, Plugin, Node Management. A teljes modell a 4. ábrán az 14. oldalon látható. A többi alrendszerről pedig a későbbiekben lesz szó, hiszen azok, már jelentős mértékben függenek a választott technológiáktól, és nem képzik szerves részét az alap logikának.

3.3.1. Az alap rendszer - modell

A BIMG alapját a Node-Slot-Parameter rendszer képezi. Paraméter lehet bármilyen tetszőleges adat (szám, szöveg, kép, vektor, mátrix stb).
Minden slotnak kötelezően két paraméterrel kell rendelkeznie, amíg az első az alapértelmezett értéket, addig a második az éppen aktuális értéket reprezentálja. A két érték típusa mindig azonos. Így látható, hogy a slotok csoportosíthatóak tárolt adattípus szempontjából, azonban ezen túlmenően működési irány alapján is rendsze-



3. ábra: A probléma ábrázolása domain modellen.

rezhetőek. Az irány azt reprezentálja, hogy az adott slot az őt tartalmazó node-ban milyen irányú kommunikációt képes végezni.

- InputSlot: Bemeneti kommunikációt végez, így bemeneti adatot reprezentál a node szempontjából. Alapértelmezetten blokkolja a node végrehajtását.
- OutputSlot: A node szempontjából kimeneti adatot reprezentál, hiszen kimeneti kommunikációt végez. A node végrehajtását nem blokkolja.

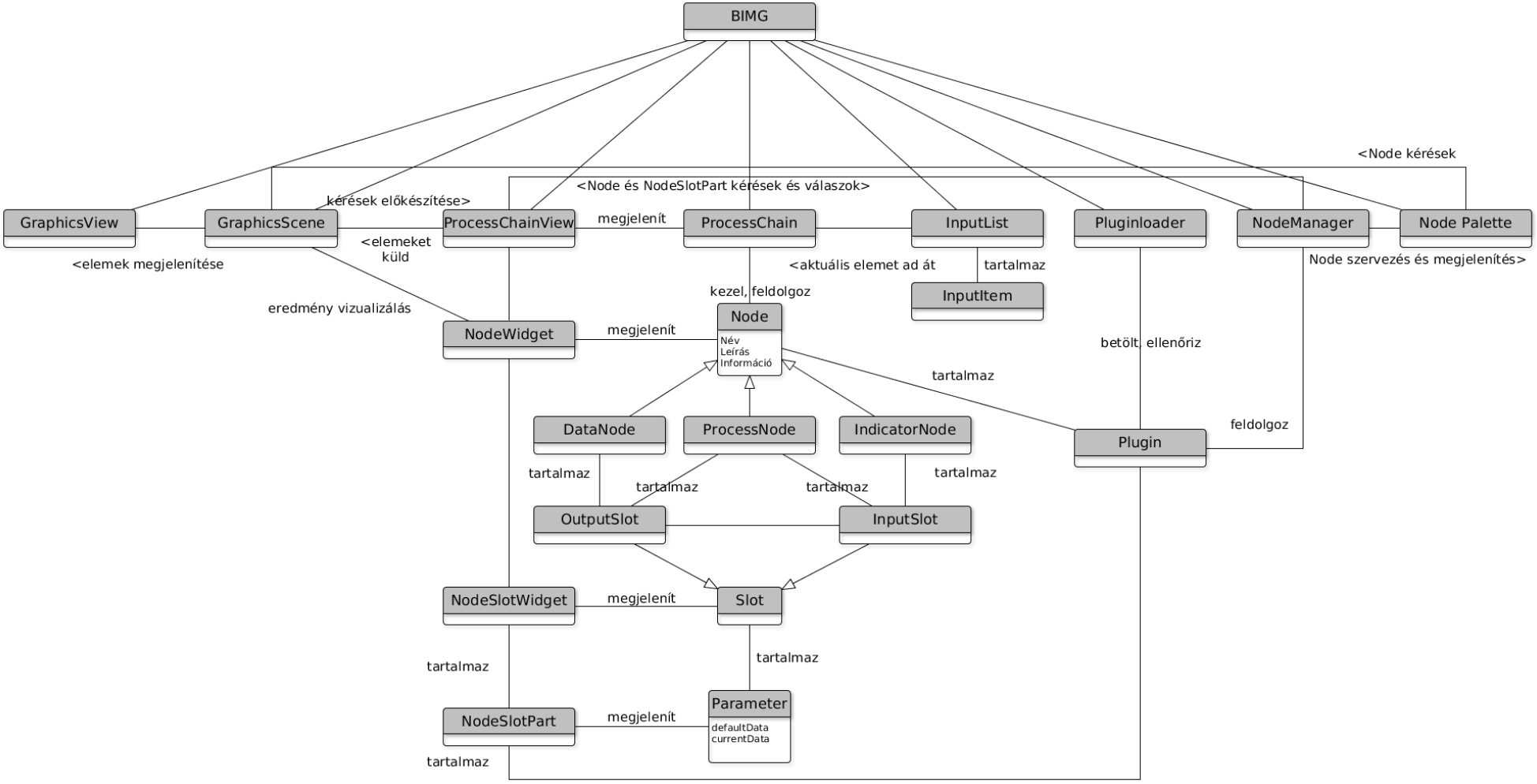
Összegezve beszélhetünk akár pl.: bemeneti számot, kimeneti képet stb kezelő slotokról. Egy slot csak egy nodehoz tartozhat, de egy node több slotot is foglalhat magában. Három féle node létezik a rendszerben:

- DataNode: Csak Output slottal rendelkezik, tehát a teljes feldolgozás szempontjából egyértelműen csak bemeneti adatforrásnak tekinthető. Mivel nincsen bemeneti slotja azonnal végrehajtható. Alapvető funkciója: adatot juttat be a ProcessChainbe.

- IndicatorNode: csak Input slottal rendelkezik, tehát a teljes feldolgozás szempontjából csak kimeneneti adatforrásként kezelhető. Alapvető funkciója: adatot juttat ki a ProcessChainból.
- ProcessNode: Input- és OutputSlotokkal is rendelkezik. Az egyetlen node típus, amely igazi feldolgozási logikával rendelkezik (tehát nem csak megjelenít, vagy vissza adértékeket hanem azokkal tényleges műveletet is végez). Végrehajtása az InputSlotok miatt alap esetben blokkolt. Alapvető funkciója: feldolgozás.

A nodek között slotok segítségével kapcsolat építhető fel. A nodeknak a végrehajtását a ProcessChain ütemezi és irányítja. Az éppen feldolgozásra szánt kép az InputListről érkezik, amelyet a ProcessChain átvesz. Az InputListben több elem is található, mindegyik egy nyers eredményképet reprezentál.
A nodek plugienkben kerülnek a rendszerbe. A plugineket a pluginmanager tölti be induláskor automatikusan, majd validálás után a plugin nyers tartalmát tovább adja a NodeManagernek. A NodeManger az átvett nyers plugin tartalmat, amely, különböző BIMG által használt objektumok listái, feldolgozza, és regisztrálja az adott node típust. Amennyiben a felhasználó a ProcessChaint egy új noddal szeretné bővíteni, a ProcessChain kérést intéz a NodeManagerhez, ami ha a kért node típus érvényes és regisztrált elkészít egy példányt, és visszadja a ProcessChain számára.

3.4. Domain modell



4. ábra: A domain modell állapota az utolsó fejlesztési iteráció végén

3.5. Választott technológiák

A feladat modellezése után döntést kellett hoznom, hogy milyen technológiákkal kívánom megvalósítani a tervezett rendszert. Több tényező is befolyásolta a döntésemet. Ezekeből két nagy csoportot írtam fel: a feladatból illetve szubjektív nézőpontból kiemelt fontosságú szempontok. A feladatból, és követelményrendszerből adódóakat a korábbi fejezetekben már részleteztem, ezért a következőkben csak a szubjektív pontokat vázolnám fel.

- Egyszerű és gyors, minőségi fejlesztés
- Könnyű dokumentálhatóság
- Legyen korábbi munkáimból rutinom az adott technológiák alkalmazásában
- Képfeldolgozási függvénykönyvtárakkal legyenek jól ellátottak a kiválasztott technológiák (nem szeretném újra feltalálni a kereket)

A program alap szerkezete Qt-val, a képfeldolgozásért felelős komponensek OpenCV-vel történő implementálása mellett döntöttem. A verziókövetést Git-el végeztem, a dokumentáció és dolgozat elkészítéséhez pedig Latex-et, Gummi-t, Doxygen-t, és Yed-et használtam.

3.5.1. Qt

Egyike a legmeghatározóbb[19] multiplatform c++-ra épülő alkalmazás keretrendszerenek. [13] Korábban már több másik projektben is sikeresen dolgoztam vele. A részletes dokumentáció és aktív felhasználói/fejlesztői bázis sokat segített a fejlesztésben. [14] [15][16] Licencelése kedvező, elérhető OpenSource és Enterprise verziója is. Olyan nagy cégek is használják mint a BlackBerry, Michelin vagy a Panasonics.[17]
A fejlesztés korai fázisa az 5.1es verzióval történt, azonban az új 5.2es verzió jelentős újításokat hozott (főként a meta-type rendszer szempontjából), ezért átálltam az 5.2.1-es verzióra.

3.5.2. OpenCV

Open Source Computer Vision Library, nyíltforrású képfeldolgozást és gépi tanulást megvalósító függvénykönyvtár.[23] Natív c++-ban implementált és erősen támaszkodik azt stl tárolókra. Sajnos gyárilag csak c/c++ és phyton-al tud hatékonyan együttműködni. Szerencsére egy vékony wrapper elkészítésével könnyen összekapcsolható Qt-val is. Funkcionalitásával széleskörű feladatok megoldására kiválóan alkalmas, technológiai lehetőségek arzenálját vonultatja fel pl.: Cuda, OpenCl. Multipaltform, még okostelefonra is elérhető a portja (Android 2010, iOS 2012). Dokumentációja is megfelelő. Jelen programban a 2.4.8-as verzióval dolgoztam.

4. Architekturális tervek és megvalósítás

A modellre építve a választott technológiák segítségével az első iterációban kidolgoztam a node-slot-parameter alrendszert.

4.1. Az alaprendszer - architektúra

4.1.1. Parameters

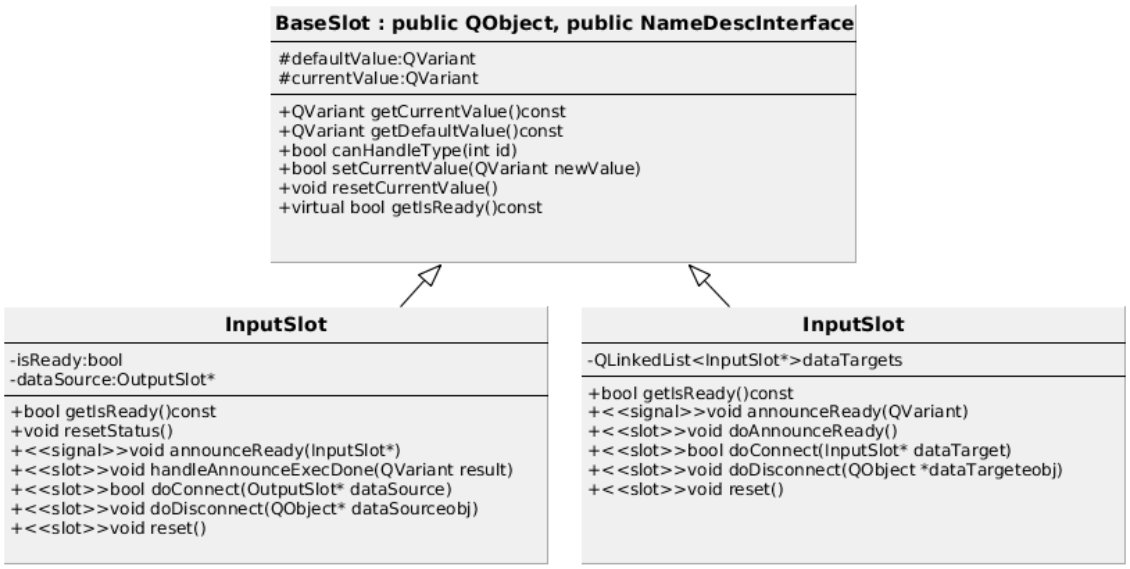
Az első tervek alapján elkészült proof of concept templates slotokra épült, ahol a template paraméterként átadott objektum volt a slotnak a paramétere. Azonban sajnos hamar be kellett látnom, hogy ez a templates implementáció nem rendelkezik elég flexibilitással ahhoz, hogy hatékonyan teljesítse a feladatokat. A legnagyobb problémát az jelentette, hogy ha egyedi plugineket és nodekat készítünk akkor a rendszernek kell tudnia kezelni az esetleges új adattípusokat is. Az adattípusok kezelése alatt nem csak az adott típusú slotok létrehozása és managelése értendő, hanem a típusok közötti megfeleltetések, konverziók is. Tehát programnak valós időben kell ezeket az adattípusokat összegyűjteni, tárolni és rendszerezni.

Erre tökéletesen alkalmas a Qtnek a Meta-Type rendszere. Bármilyen tetszőleges osztályt metatípusként deklarálhatunk, így lehetőségünk van az adott objektumokat QVariantban eltárolni. [24] Mivel a metatípus azonosítók kiosztása, és feloldása valósidőben történik, így megoldható a pluginekben található osztály deffiníciók a főprogramban történő használata.

A QVariant alapvető adattípusok számára biztosít konverziós lehetőséget. Az 5.2-es verziótól már arra is van lehetőségünk, hogy mi deffiniáljunk különböző metatípusok közötti konverziós műveleteket.
Az alábbi lehetőségeket összegezve egyértelmű volt a döntés, hogy a slot paraméterek a qt metatype rendszerre épülve kerülnek megvalósításra. (Az adattípusok betöltését és konverziók regisztrálását a pluginek működéséről szóló részben fejtem ki.)

4.1.2. Slots

A korábban említett templates architektúra elhagyása lehetővé tette, hogy a nodek slotjai is QObject alapokra épüljenek. Tehát a kommunikációjuk és így az adat továbbítás a nodek között a qt Signal/Slot mechanizmusára épülhessen.



5. ábra: A node slotok osztálydiagramja

A 9. ábrán látható a slotok jelenlegi osztálydiagramja. A funkcionalitások szervezésekor (hogy mi kerüljön a BaseSlot osztályba és mi a gyermek slot osztályokba), a logika az volt, hogy a BaseSlot gyakorlatilag csak és kizárólag az aktuális és az alapértelmezett paraméterek kezelését valósítsa meg, míg az input és output slot pedig a paraméterek fölé biztosít egy réteget ami kezeli a slot kapcsolódásokat, és

állapotokat. Ennek a döntésnek az oka annyi, hogy két fő szabályt osztály architekturális szinten kívántam megkötni:

- Egy input slothoz csak egy output slot kapcsolódhat.
- Egy output slot több input slotba is szolgáltathat adatot.

Tehát nem a doConnect() és a doDisconnect() függvények működése határozza meg a kapcsolódási mechanizmust, hanem az adott slot típus ne is tudjon más számú és típusú kapcsolatot létesíteni. Tény, hogy ez eléggé erős megkötés, de a feldolgozás és nodek kiterjesztési sorrendje nagyon erőteljesen támaszkodik erre a két szabályra, így vált indokolttá ennek a döntésnek a meghozatala.

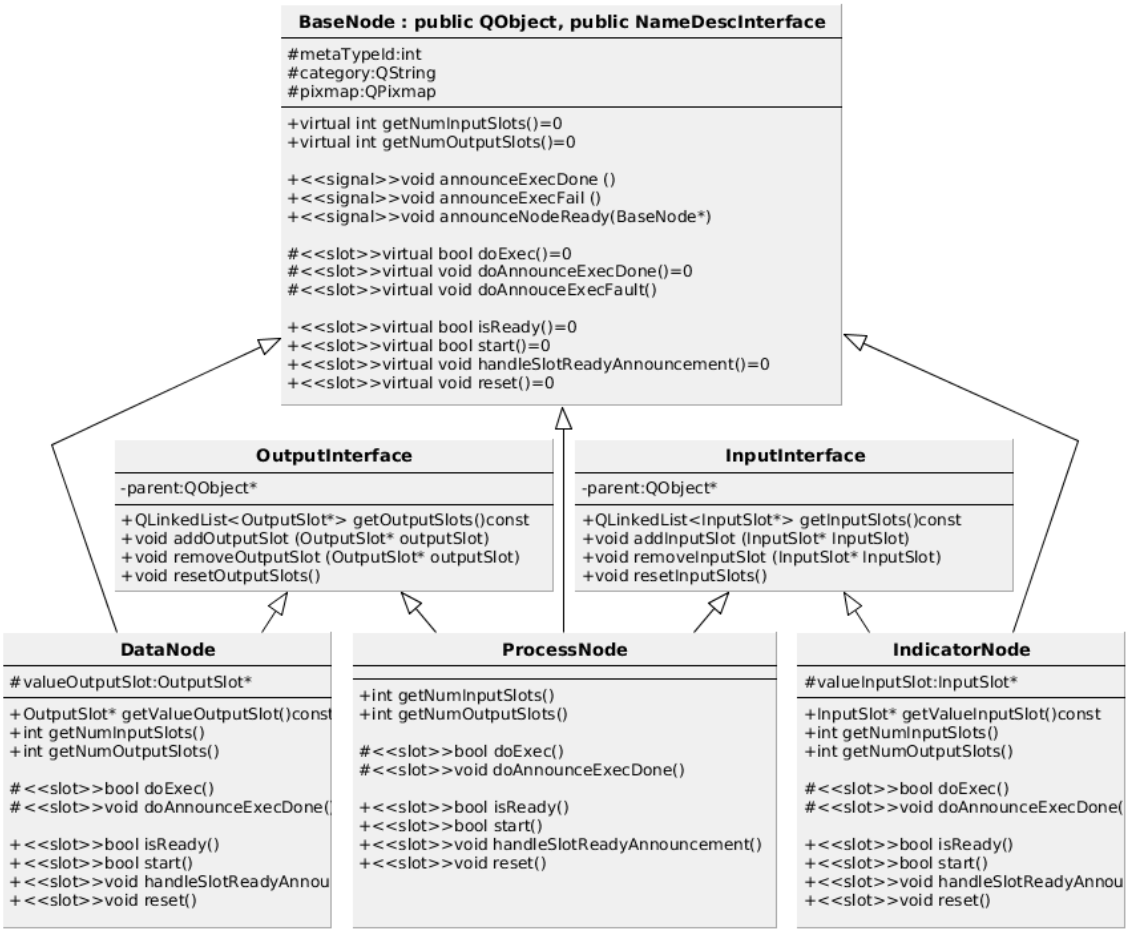
A kapcsolat mindig az inputslot irányából jön létre. Tehát amikor az outputslotnál kezdeményezzük a kapcsolat létrehozását az átadja a működést az cél inputslotnak. Ennek egyik oka a fent említett megszorítások: azaz inputslot felől szigorúbb a kapcsolat létrehozása, a másik oka, hogy így a kapcsolatot kezelő kódot csak egy helyen kell karbantartani és a későbbi esetleges fejlesztések nem okozhatnak inkonzisztensz kapcsolódási állapotot.

A slotok közötti adatáramlás iránya outputslot - inputslot irányú. Az adatok átadása az outputslot által emitált announceReady(QVariant) signállal kezdődik, és az inputslot handleAnnounceExecDone(QVariant) függvényének meghívásával zárul. A két slot közötti kapcsolat a Qt signal-slot mechanikájával történik. Ez a működés megfigyelhető a 7. ábrán.

4.1.3. Nodes

Már a slotok osztálydiagrammján is látható, hogy az alaposztály egy NameDescInterface osztálynak is a gyermekosztálya. Ez a kis interfész jellegű osztály általános adattagokat (és getter és setter függvényeket) biztosít bármely gyermek osztályának. Itt jellemzően a felhasználó munkáját könnyítő információk kerülnek tárolásra: az objektum olvasható neve, rövid leírása, és súgó szöveg.

Bár az osztály diagrammon nem látható egyértelműen, de a slotok BaseNode szinten tárolódnak. Mivel minden slot és node kötelezően QObject ezért kiválóan



6. ábra: A node slotok osztálydiagramja

használhatóak a gyermek szülő funkciók. Ezek QObject szinten vannak implementálva. Használatuk jelentős előnyökkel jár pl.: a node átveszi a slot feletti irányítást: tehát amikor megszűnik a node automatikusan felszabadításra kerül az összes slot. A slot felszabadításakor emitálásra kerül egy destroyed(QObject*) signal, amely az adott slot doDisconnect(QObject*) slotjával van összekötve. Ezzel a struktúrával elértem, hogy egy node törlésekor a gyermek slotjainak az összes kapcsolata automatikusan felbontásra kerüljön. Ennek előnye, hogy nem maradhatnak érvénytelen kapcsolatok a rendszerben, amelyek már nem létező slotra mutatnak.

A BaseNode osztály a slotok tárolásán és alacsony szintű kezelésén túl, deffiniál egy sor virtuális függvényt, melyekkel a magasabb szintű node kezelő és feldolgozó mechanizmusok (ProcessChain) számára biztosít interfészt. Az alapvető működésük, és a slotokkal történő interakciójuk megfigyelhető a 7. ábrán. Ezek közül a legfontosabb függvények működése:

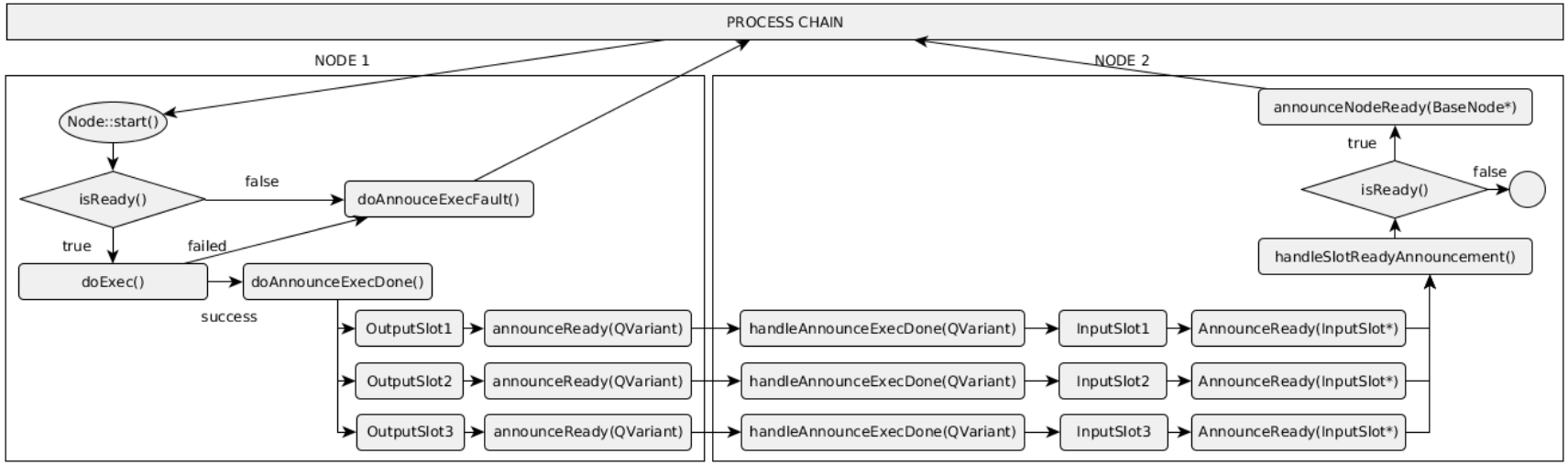
- start(): Ez minden node belépési pontja a ProcessChain ezt a függvényt hívja meg végrehajtáskor, ha a node készen áll a végrehajtásra akkor lefut a benne található logika. A futás eredményét (sikeres vagy sikertelen) visszatérési értéke és az emitált signalok mutatják.
- reset(): Alapállapotba állítja vissza a nodet. Erre akkor van szükség amikor a ProcessChain egy új bemeneti adatot szeretne feldolgozni. Ilyenkor a node és az összes slotja felveszi az alapértelmezett állapotot.
- isReady(): Ellenőrzi a node inputslotjainak az állapotát, ha mindegyik slot készen áll a végrehajtásra (tehát már mindegyik input slot kapott bemeneti adatot), akkor igaz értéket ad vissza: tehát a node végrehajtható.
- handleSlotReadyAnnouncement(): Amikor bármelyik input slot ready állapotba vált akkor, ellenőrzi a teljes node állapotát (isReady()) ha végrehajtásra kész, akkor meghívja announceNodeReady(BaseNode*) függvényt
- announceNodeReady(BaseNode*): Jelzi a ProcessChain számára ha végrehajtásra kész állapotba került a node.
- doAnnounceExecDone() és doAnnouceExecFault(): a node végrehajtásának eredményét jelentik be.
- doExec(): Itt található a node feldolgozó logikája. Amennyiben valaki új nodet fejleszt ezt a függvényt kell felül deffiniálnia.

Mivel a BaseNode osztály tervezésekor az volt a cél, hogy slot típustól függetlenül működjön ezért két segéd osztályt kellett deffiniálni. Ezek az OutputInterface és az InputInterface. Feladatuk egyszerű: egy szűlőként kapott QObject-ből visszadják az összes számukra kezelhető slot típust. Továbbá interfészt biztosítanak, hogy új slotokat adjunk a nodehoz és a régieket eltávolítsuk. Ha az adott interface osztályból örököltetünk akkor megkapjuk a lehetőséget, hogy az adott slot típusokkal dolgozzunk. Amennyiben a későbbiekben egy másik típusú slotot szeretnék elhelyezni a rendszerben pl.: ami csak konfigurációs segéd adatokat biztosít a node számára, akkor értelemszerűen készíteni kell egy interface osztályt, amiből ha örököltetünk már is rendelkezésre állnak az új típusú slotok.
Az osztálydiagrammon látható örököltetésekből egyértelműen kiderül, hogy melyik node típus milyen slotokkal képes dolgozni. A DataNode csak outputot, az

IndicatorNode csak inputot, a ProcessNode pedig mind a kettőt képes használni. A módszer előnye, hogy így kis apró szerepkörökből hamar össze lehet állítani a node működési területét, majd a BaseNode virtuális függvényei kifejtésével meghatározható a pontos működés.

4.1.4. Node-Slot adatáramlás

Lenti ábrán megfigyelhető a nodek végrehajtási mechanizmusa során keletkezett, információk tovaterjedése a kapcsolódott nodekkal, továbbá a legalapvetőbb ProcessChain és node interakciók. (Működésükről bővebben a Nodes, és Slots alfejezetkben lehet olvasni.)



7. ábra: A Node-Slot adatáramlás

4.1.5. ProcessChain

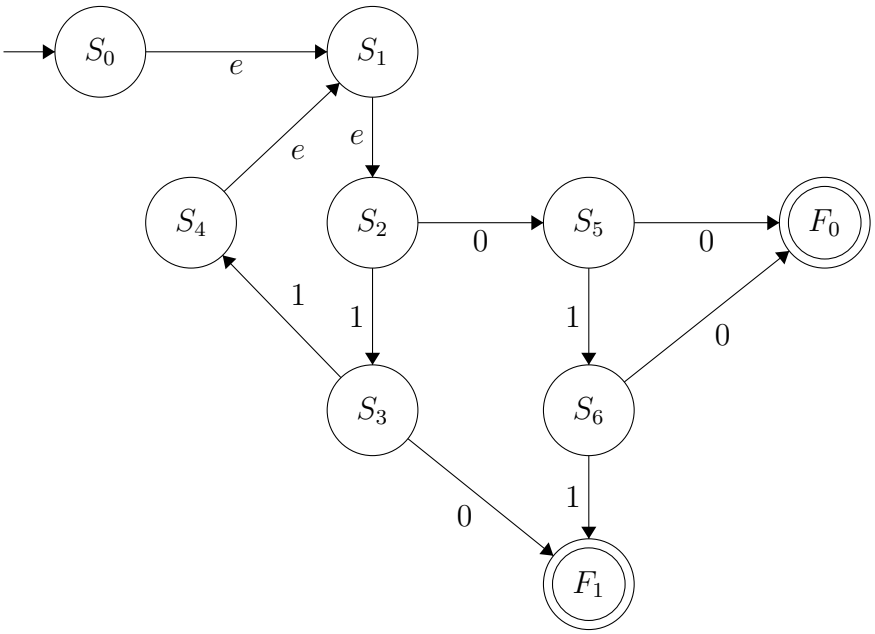
A korábbi pontokból látható, hogy a Node-Slot-Parameter alrendszerrel gyakorlatilag tetszőleges feldolgozási sor összeállítása lehetséges. Ez a feldolgozási sor tekinthető egy irányított gráfnak. Így az alábbi megfeleltetések jönnek létre:

- csúcsok: a feldolgozási sor nodejai,
- élek: nodek slotjai közötti kapcsolatok,
- élek iránya: a slotok kapcsolati irányának felel meg. (Outputslotból mutat inputslot felé.)

Az alábbi logika mentén tovább haladva a feldolgozási sort, tehát ahogy a nodeknak az egymás után történő sorrendhelyes végrehajtását tekinthetjük a gráf egy bejárásnak.
A gráf bejárására lehetőségünk van egy állapotgépet felírni. Az állapotgép jellemzői:

- Három darab memória területe van. L_0, L_1 és L_2
  - A memória területek között a nodekat átmozgatjuk, nem másoljuk: tehát egy node mindig csak egy területen szerepelhet
  - L_0 induláskor minden node itt található
  - L_1-ben tárolunk minden ready állapotú nodet (amelyek készen állnak a végrehajtásra)
  - L_2-ba kerülnek a már kiterjesztett nodek (itt jelenik meg a kiterjesztési sor)
- Az állapotgép minden állapotában vagy egy művelet történik, a művelet eredménye lehet sikeres (1), és sikertelen (0), egy állapotból továbbléphetünk annak eredményétől függetlenül is. Tehát az abc: {0,1,e}
- Az állapotgép három esetben állhat meg:
  - Az összes node kiterjesztése sikeres volt: ez esetben a művelet egyértelműen sikeres volt.

– Nincsen további kiterjesztésre kész node. Ez esetben ha az L_0-án nem maradt egy darab IndicatorNode sem, akkor a feldolgozást sikeres. (Hiszen előfordulhatnak izolált csúcsok a gráfban.) Azonban ha található a még nem kiterjesztett nodek között (L_0) indicator node a feldolgozás sikertelen volt.
– Ha valamelyik node kiterjesztése sikertelen volt. (A kiterjesztés közben hibalépett fel. Ezt többnyire a node nem megfelelően elkészített doExec() metódusában bekövetkezett hibát jelent.)



Állapot	Művelet
S_0	Kezdő állapot.
S_1	L_0-án található összes ready állapotú átmásolásra kerül L_1-re.
S_2	Található elem L_1-en?
S_3	L_1 első eleme kiterjesztésre kerül. Sikeres volt?
S_4	A sikeresen kiterjesztett eleme L_1-ről átmozgatásra kerül L_2-re.
S_5	Található node L_0-án?
S_6	Az L_0-ás nodek között található IndicatorNode?
F_0	A feldolgozás végetért! Eredmény: SIKERES FELDOLGOZÁS!
F_1	A feldolgozás végetért! Eredmény: SIKERTELEN FELDOLGOZÁS!

1.4. táblázat: Az állapotgép állapotainak értelmezése

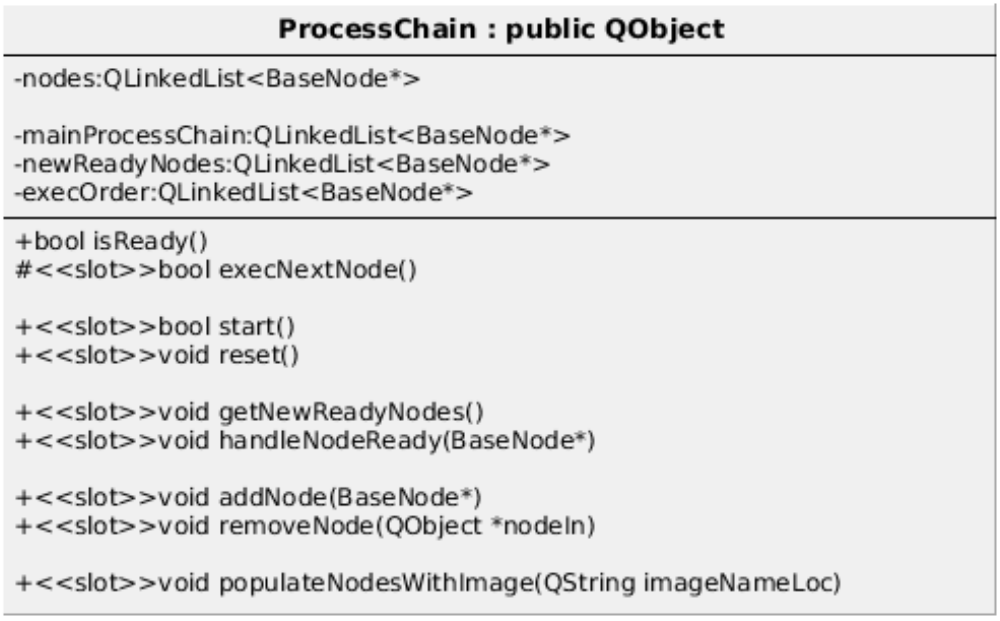
Látható, hogy mennyire előnyös ez az állapotgépes feldolgozás, hiszen nem kell a gráfban kört keresni illetve egyéb más módon elemezni azt. Adott pár jól deffiniált művelet és szabály ami alapján egyértelműen végrehajtható a felhasználó által

felépített feldolgozási sor. Az adatok áramlása és a nodek ready állapotba kerülése pedig alacsonyabb szinten teljes mértékben rendezésre kerül és a node-k elfedik azt a ProcessChain elől.

A korábbiakban bemutattam, hogy egy bemeneti elemre, hogyan működik a feldolgozás. Természetesen van lehetőség több elem egymás utáni feldolgozására is. Ennek feltétele, hogy miután egy adott bemeneti elem sikeresen feldolgozásra került exportáljuk az eredményt, majd alaphelyzetbe állítsuk a ProcessChaint és a benne található nodekat. A be és kimeneti adatokat az InputList biztosítja a ProcessChain számára. (Amelynek beállítása a populateNodesWithImage() függvényben történik.)

Megjegyzés: ha valamelyik bemeneti elem feldolgozása sikertelen volt akkor a process chain ugyanúgy alapállapotba kerül vissza (a hibaüzenetek nyugtázást követően).

Az alábbi osztálydiagrammon látható a ProcessChain legfontosabb adatagjai és függvényei.



8. ábra: ProcessChain osztálydiagramja

Megfigyelhető a nodek általános tárolója (nodes), és L_0 (mainProcessChain), L_1 (newReadyNodes) és L_2 (execOrder) nevű láncolt listák. (Az állapotok műveletei

legnagyobb részt az execNextNode() függvényen belül találhatóak.)

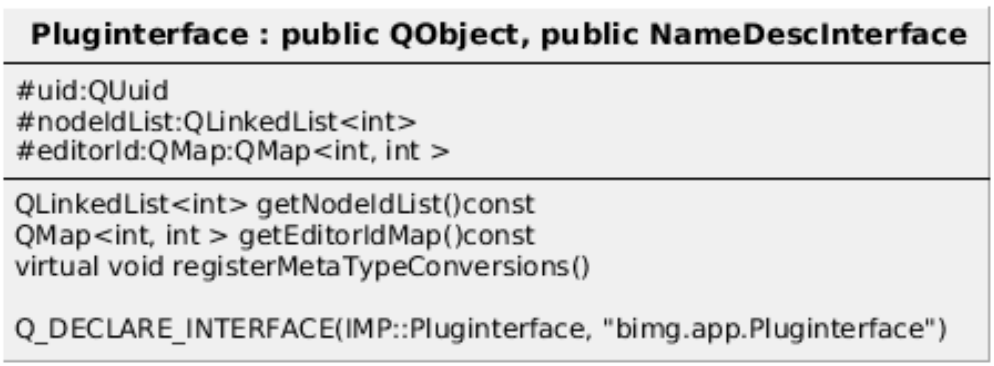
4.2. Pluginek

Az alap Node-Slot-Parameter rendszer után a második legnagyobb modulnak a plugin rendszer tekinthető. Legfontosabb feladata, hogy a BIMGt plusz funkcionalitásokkal, adattípusokkal töltse fel. A tervezés és a fejlesztés kettő pont köré csoportosult:

- Egy jól deffiniált kompakt, de flexibilis interfészt kell kialakítani amely nagy verzió ugrások után is változatlan marad.
- Nagy mennyiségű Data-, Process- és IndicatorNodet és ezekhez tartozó adattípusokat és konverziós megfeleltetéseket kell áthordozni a pluginból a főprogramba.

4.2.1. Pluginterface

Kiindulási alapnak a Qt által biztosított plugin rendszert használtam. A rendszer jól működő és bevált, hiszen a Qt is erősen építkezik erre a rendszerre[26]. A BIMGben az alacsonyabb szintű plugin api-t használtam.



9. ábra: A pluginterface osztálydiagramja

Első lépésként deffiniáltam egy plugin interface osztályt, amely a Pluginterface nevet kapta. Azon túl, hogy a Q_DECLARE_INTERFACE() makró segítséggel egy

interfészt deklarálása történik benne még három fő funkcionalitást különíthetünk el benne.

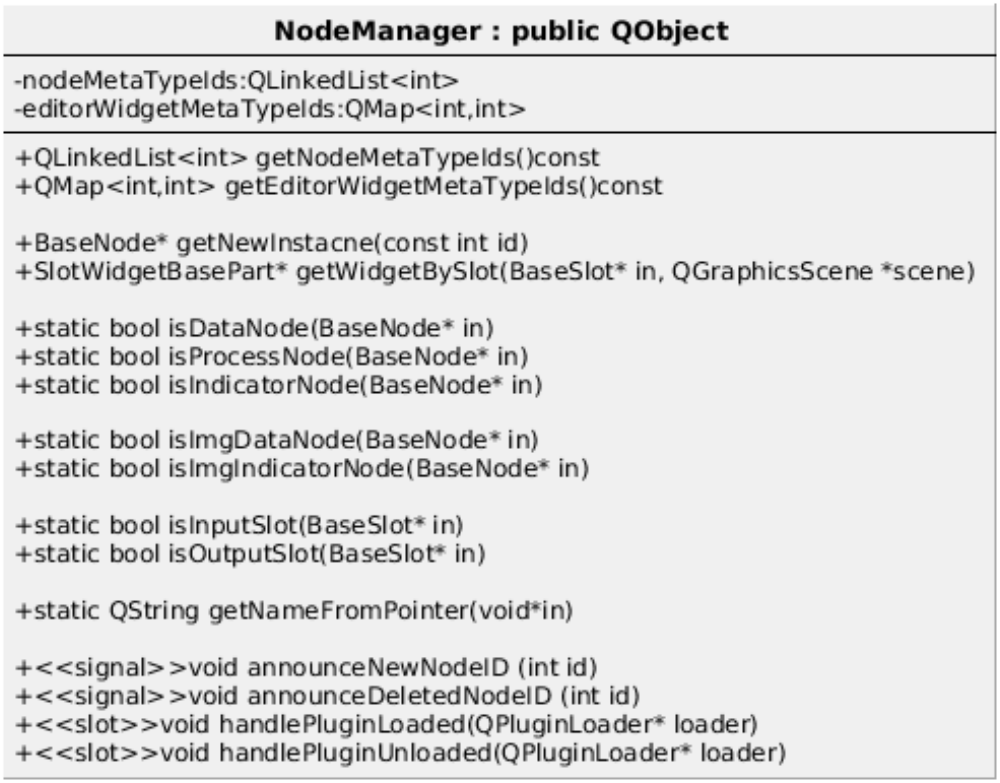
- Tartalmaz egy UUID[27] adattagot. Amely egy 128 bites állandó és egyedi azonosítója a pluginnak. A jelenlegi architektúrában ez a változó nem használt, azonban a további fejlesztési lehetőségek részben található PluginContainer és PluginUpdater rendszerek megkövetelik használatát. Ugyan ez érvényes a NameDescInterface által biztosított adattagokra is.
- Adott két előkészített tároló nodeIdList, és editorIdMap ahol különböző metatype-idk tárolására van lehetőség. (Konkrétan a Data-, Process- és IndicatorNodekhoz továbbá a SlotWidgetBasePartokhoz tartozó metatype-idkre.)
- Egy előkészített függvény: registerMetaTypeConversions(), amelyben deffiniálhatjuk a különböző saját fejlesztésű adattagok közötti konverziós lehetőségeket.

4.2.2. Plugin metaobjektumok és ellenőrzések

Ahogy fentebb említettem, több objektum metatípus leírója utazik a pluginnel. Gyakorlatilag ha valamelyik tárolóban eltárolunk metatype-id az betöltéskor beolvasásra kerül. Természetesen nem ellenőrzés nélkül, a minden plugin betöltéskor két szintű ellenőrzésen megy keresztül:

- A PluginLoader ellenőrzi, hogy egy szabványos pluginről van-e szó. Tehát Pluginterface ős osztállyal és interfész deklarációval továbbá Q_OBJECT makróval kell rendelkeznie (QObject osztályból örököltetni nem szükséges, hiszen a Pluginterface eleve a QObject osztály gyerekosztálya.) amint ez sikerel lezárul a PluginLoader osztály létrehoz egy QPluginLoader pointert, amelyet a pluginLoaded signállal megküld a NodeManager osztálynak.
- A második szintű ellenőrzést a NodeManager::handlePluginLoaded(...) végzi. Végig iterál a beregisztrált azonosítókon. Tárolási helytől függően, vagy BaseNode, vagy SlotWidgetBasePart objektumot próbál készíteni a metatype rendszer segítségével. Amennyiben ezek valóban megfelelő típusú objektumoknak bizonyulnak beregisztrálja a saját tárolójába. A metatype-id konverziós definíciók regisztrálása is itt történik.

4.2.3. NodeManager



10. ábra: A nodemanager osztálydiagramja

A NodeManager eléggé fontos architekturális szerepet tölt be a BIMG rendszerben. Ez az osztály tekinthető a pluginek és a BIMG közötti határvonalnak. Itt történik (a korábban már részletezett módon) a pluginok tartalmának beolvasása és ellenőrzése. Továbbá itt van lehetőségünk új példányokat igényelni ezekből az objektumokból, paraméterként átadott metatype id segítségével. A NodeManager csak azokból az objektumokból készít új példányt, amelyek regisztrálva vannak nála. Ennek előnye, hogy ilyen módon letilthatóvá válnak bizonyos pluginek vagy pluginrészek használata.

Például a későbbiekben tárgyalt NodePalette, először kikéri a regisztrált és érvényes metatípus-idk listáját (getNodeMetaTypeIds()), majd a getNewInstacne(...) segítségével kér új példányokat az adott típusú objektumból.

4.2.4. Egy példa plugin

A plugin rendszer bemutatátás egy példával zárnám, ahol egy plugin létrehozásának menetét ismertetem:

1. Hozzunk létre egy szabványos C++ dinamikusű függvénykönyvtárat, majd származtassuk le a Pluginterface ősosztályból, végül alkalmazzuk rajta a Qts interfész makrókat.

```
//plugin_alpha.h
class PLUGIN_ALPHASHARED_EXPORT plugin_alpha : public ↩
    Pluginterface
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID "org.examples.MyInterface")
    Q_INTERFACES(IMP::Pluginterface)

    //...
};
```

2. Készítsük el a saját nodejainkat, és adattípus konverzióinkat. Nodek esetén használjuk ősosztálynak valamelyik node típust (DataNode, ProcessNode vagy IndicatorNode). A típus konverzióknak a forrás objektum típusból egy const refferenciát kell átvennie, és egy cél objektumot kell visszadnia

```
//mat2image.h
Q_DECLARE_METATYPE(cv::Mat)
class Mat2Image{
public:
    static cv::Mat castImgToMat(const QImage& image);
    static QImage castMatToImg(const cv::Mat& mat);
```

   A fenti példában az OpenCV Mat objektum típuskonverziója történik meg QImage típusra és vissza. Megjegyzés: bármilyen külsős libbel végezzük a képfeldolgozást ajánlott a felhasznált adatformátumot ilyen módon regisztrálni.

3. A plugin konstruktorában regisztráljuk a metatípusokat! Illetve deffiniáljuk felül a registerMetaTypeConversions függvényt.

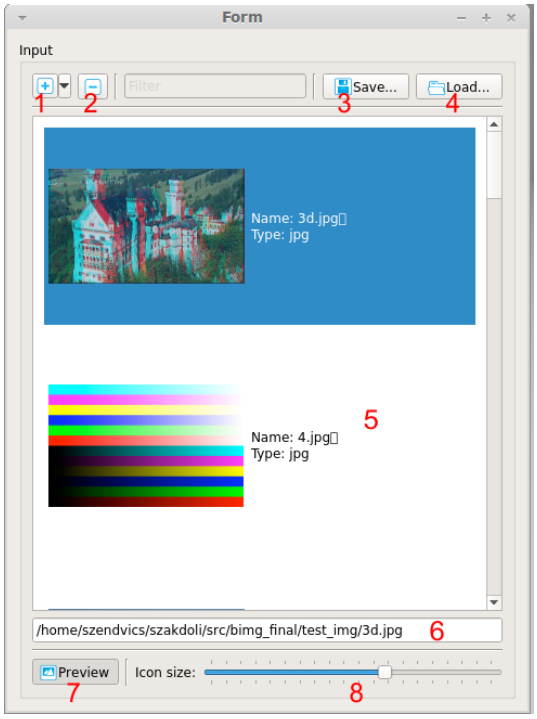
```
//plugin_alpha.cpp
Q_DECLARE_METATYPE(RGBASplitter)
Q_DECLARE_METATYPE(RGBAJoin)

plugin_alpha::plugin_alpha():
    Pluginterface(QUuid::createUuid(),"PLUGIN_ALPHA","CORE␣↩
        FEATURES")
{
    //data
    this->nodeIdList<<qRegisterMetaType<DoubleDataNode>();
    this->nodeIdList<<qRegisterMetaType<StringDataNode>();
    //..
}
void plugin_alpha::registerMetaTypeConversions(){
    QMetaType::registerConverter<cv::Mat,QImage>(Mat2Image↩
        ::castMatToImg);
    QMetaType::registerConverter<QImage,cv::Mat>(Mat2Image↩
        ::castImgToMat);
}
```

4.3. InputList

Az InputList az egyik legvékonyabb architekturális elem a BIMGben. Feladata roppantul egyszerű: lehetőséget ad a felhasználó számára, hogy összeállítson egy bemeneti listát, amelynek elemeit átathatja a ProcessChain számára feldolgozásra. Lehetőségünk van egyszerre egy vagy több elemet a listhoz hozzáadni, vagy eltávolítani róla. Továbbá előnézeti lehetőséget biztosít a felhasználó számára aki így feldoglozásra küldés előtt megvizsgálhatja a nyers képet. Architekturális szempontból a currentPicSelectionChanged (QString) signaljának van a legnagyobb jelentősége, amely a ProcessChain számára küldi meg az éppen aktuális bemenet elérését.

4.3.1. Input műveletek



11. ábra: Input műveleti ablakban elérhető lehetőségek. 1: Egy vagy több kép hozzáadása az input listához. 2: Aktuális kép törlése az input listáról. 3: Aktuális inputlista mentése. 4: Korábban mentett inputlista betöltése. 5: Aktuális input lista vizsgálati helye. 6: Inputok gyors előnézeti területe. 7: Gyors előnézeti mód ki/be kapcsolása. 8: Gyors előnézeti mód méretének változtatása

4.4. GUI

Az eddig tárgyalt modulok (az InputList kivételével) és alrendszerek nem rendelkeztek se grafikus sem más féle felhasználói felülettel. Ezeknek a moduloknak a megfelelő interfészei használatával készítettem a grafikus felhasználói felületet. A GUI munkálatok kettő fontos részből álltak, az első részben a Qt által biztosított osztályokat bővítettem ki a projekt szempontjából szükséges funkcionalitásokkal. A másodikban ezekre támaszkodva készítettem saját fejlesztésű GUI elemeket, hiszen nem volt megfelelő gyári GUI elem ami képes lett volna megjeleníteni a DataNode, ProcessNode vagy IndicatorNode objektumokat.

4.4.1. Scene&View - guiview lib

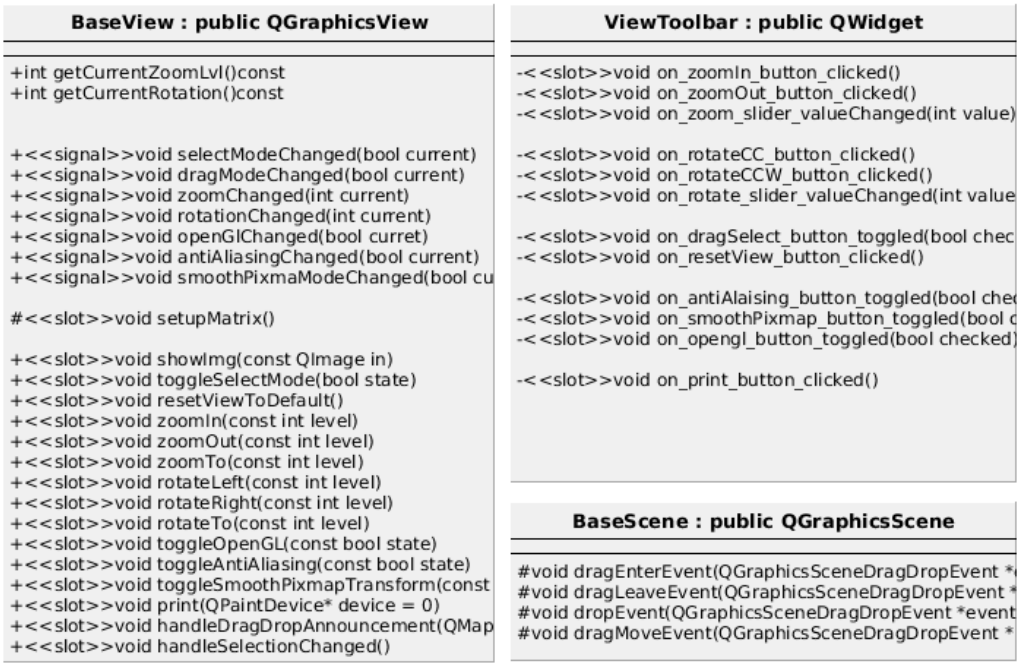
Architekturális szempontból a GUI peremén (a userhez a legközelebb), helyezkednek el a QGraphicsView és QGraphicsSenere gyermekosztályai.

12. ábra: A Base view tool barja. 1: Nagytás, 2: Kicsinyítés, 3: Méretezés értékhez, 4: Méretezés értékhez, 5. Forgatás jobbra, 6. Forgatás balra, 7: Forgatás adott értékhez, 8: Forgatás adott értékhez, 9: Interakciós mód, 10: Nézeti ablak beállításainak nullázása, 11: AAA engedélyezése/tiltása, 12: Simított bitkép tranzformációk tiltása/engedélyezése , 13: OpenGL renderelés engedélyezése/tiltása, 14: Nézeti ablak jelenlegi tartalmának mentése

A BaseView osztály egy Toolbarral osztállyal párhuzamosan fejlesztve készült el. Segítségükkel a felhasználó alap interakciókra képes: zoomolni, forgatni a megjelenített grafikát a képen, adott objektumokat kijelölni és mozgatni, különböző renderelési opciók ki és be kapcsolni, és az éppen aktuális megjelenítési állapotokról készült képet elmenteni. A BaseScene kapja el a Drag&Drop és egyéb eseményeket, erre az osztályra építve készült el a NodeEditorScene. (További részletek a 13. ábráról olvashatóak le.) Ezeknek a kis osztályoknak a tervezésénél és implementációjánál kifejezetten arra törekedtem, hogy ne kerüljön ide a BIMGhez köthető funkcionalitás, adattag stb, így jelenleg egy külön statikus libként van a BIMGbe bele forgatva guiview néven.

4.4.2. NodePalette/NodePaletteForm

A NodePalette biztosítja a felhasználó számára, hogy a BIMG rendszerben található nodekat használhassa, illetve bővebb információt szerezzen róluk. A nodek a könnyebb rendszerezhetőség érdekében füleken jelennek meg. A csoportosítás alapja a BaseNode::category (QString) változónak a tartalma, amelyet az adott node konstruktorában adhat meg a fejlesztő. (Ugyan ezen a módon deffiniálható a node, neve, leírása, súgó szövege és mini-iconja.). A Nodehoz tartozó súgás jelenleg egy jobb kattintással hozható elő, új node létrehozás rendkívül egyszerű: hiszen elég az NodePalettán található elemet a közismert fogd és vidd módszerrel a NodeEditor felületére mozgatni.

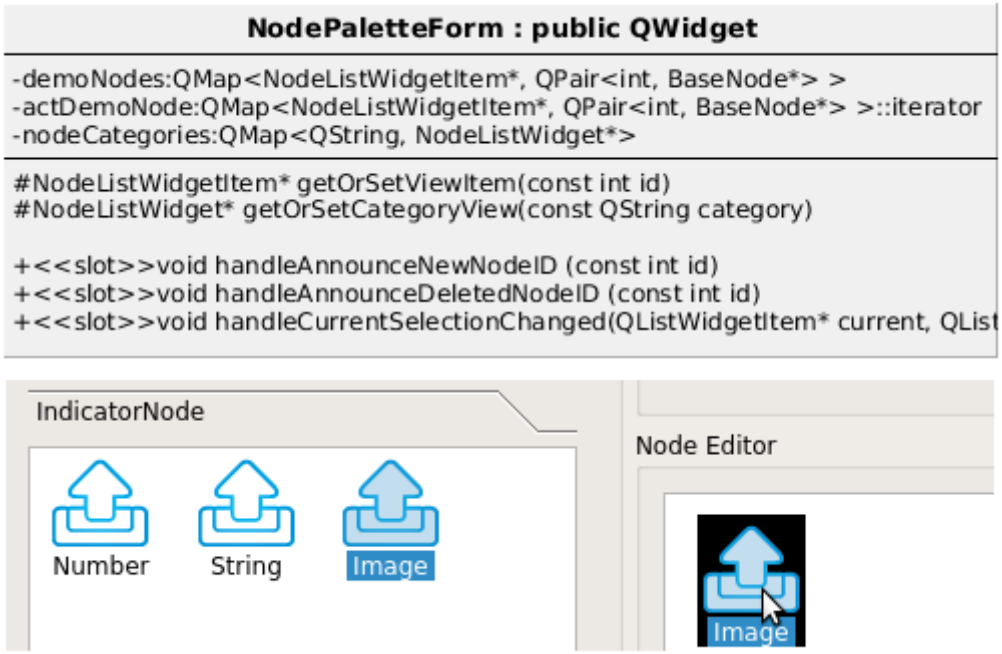


13. ábra: A ViewGui libnek az osztálydiagramja

Architektúrális szempontból a NodePalette (NodePaletteForm), gyakorlatilag a NodeManager részére képez egy viewet. Követlen értesítést kap (announceNewNodeID (int id), announceDeletedNodeID (int id) signáljai a NodeMangernek) a frissen regisztrált, és az éppen letiltott metatype-idkről. A handleAnnounceNewNodeID(int) és handleAnnounceDeletedNodeID(int) slotokkal kezeli ezeket az értesítéseket. Minden egyes metatype-idhez kér a palette egy demo node példányt a managertől. Ezek a nodek, csak arra szolgálnak, hogy a NodePalette információt tudjon szolgáltatni a felhasználó részére a noderől. (Tehát feldolgozási sorhoz nem lesznek hozzáadva stb.)

4.4.3. NodeEditor/NodeEditorScene

A megjelenítési műveletek eseménykezelő és elem mappoló színhelye. A ViewGUI::BaseScene osztálynak a gyermekosztálya. Fogadja a drag&drop illetve egyéb más felhasználói interakciókat, minimális logikával rendelkezik: gyakorlatilag megszűri az eseményeket. Minden lehetséges BIMG szempontból érdekes eseményt továbbít a ProcessChainView, részére. pl.: Ha elfog egy drop műveletet akkor ellenőrzi, hogy az adott objektum MIME típusa érdekes lehet-e a BIMG számára.



14. ábra: A NodePalette osztálydiagramja, és működés közbeni képernyő fotója

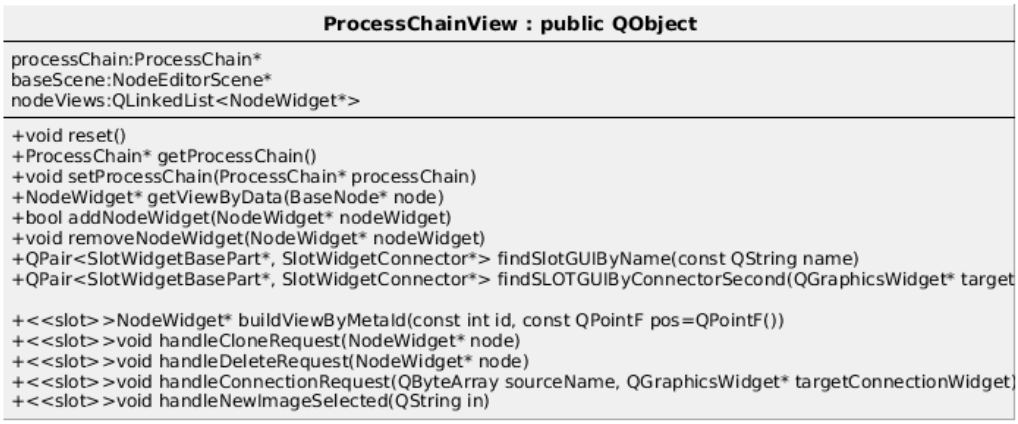
Jelenleg ilyen figyelt metatípusok a: "application/x-qabstractitemmodeldatalist" ha tartalmaz Node metatype-idt (ez a Qt::UserRole+7462-es role), a "application/x-bimgnodeconnector" amely akkor érvényes ha tartalmaz "sourceobj" Stringgel indexel adatot. Az első mimeadat a NodePaletterről történő fogd és vidd művelet eredménye a második két NodeWidgetSlotConnection közötti kapcsolat létesítési kísérlet eredménye.

4.4.4. ProcessChainView

Ha a Node-Slot-Parameter rendszer feldolgozó logikája a ProcessChain volt akkor várható, hogy a ProcessChainView is hasonló architekturális feladatokat tölt be (csak a view oldalon). Ez az állítás helytálló. A következő listában röviden összegzem a feladatokat, a lentebb található osztálydiagrammon pedig megtalálhatóak ezek a feladatokhoz megfeleltethető függvények és adattagok.

- Itt tárolódik és konfigurálódik a NodeWidgeteket. (pl: itt kerül beállításra a NodeEditorScene elérése stb.)
- Fogadásra és feldolgozásra kerülnek a NodeWidgetek különböző kérelmeit: klónozás, és törlés.

- Fogadásra kerül a NodeEditorScenetől érkezett és előfeldolgozott eseményeket. Az események-NodeWidgetek kimappolása és ellenőrzése is itt történik pl. 2 Slot összekapcsolási kérelme esetén
- Itt történik az információ csere a ProcessChainnel, a feldolgozás jelenlegi állapotáról, továbbá az új és törölt (illetve hozzáadni és törölni kívánt) nodekről.



15. ábra: A ProcessChainView osztálydiagramja

4.4.5. NodeWidget

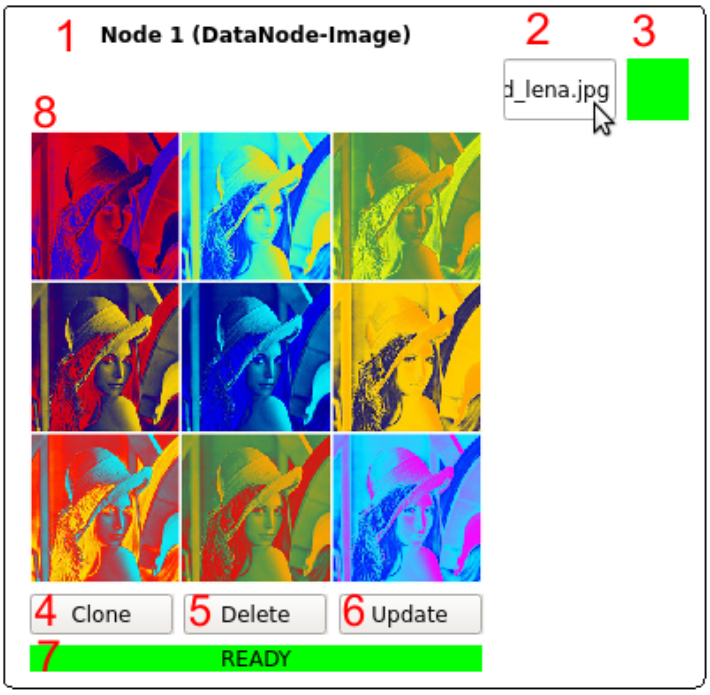
A GUI fejlesztésben a NodeWidget alrendszer tervezése és implementálása jelenetette a legnagyobb kihívást. A NodeWidget a QGraphicsWidget osztályra lett felépítve. A widget több rész widgetre bontható fel:

- NodeWidgetBasePartokra amelyek közvetlenül a BaseNode valamely elemét jelenítik meg
- SlotWidgetBasePartokra amelyek közvetlenül egy darab slot megjelenítését és kezelését végzik

A NodeWidget azontúl, hogy levezényli a hozzárendelt BaseNode megjelenítését (az alap adatok vizualizálásán túl, az esetleges slotok megjelenítését és kezelését). Különböző kéréseket is küldhet a ProcessChainViewnek. Jelen implementáció alapján kettő kérés érkezhet egy NodeWidgettől: az egyik a clone, azaz hogy egy vele

azonos típusú NodeWidget kerüljön fel a megjelenítőre (és Node letárolásra a ProcessChainben); a másik a delete, azaz törlés tehát amikor a NodeWidget sajátmaga eltávolítását kérvényezi a ProcessChainViewből és ProcessChainból.

A NodeWidget tartalmának elrendezése és kirajzolása egy GridLayout segítségével került megvalósításra.



16. ábra: Egy image DataNode. 1: A Node neve. 2: A DataNode output slot szerkesztésének helye (jobb klikkel további funkciók érhetőek el.). 3. Node isReady staus indicator. 4. Klónozás 5. Törlés. 6. Frissítés 7. Node státusz állapota. 8. Kisméretű előnézeti kép.

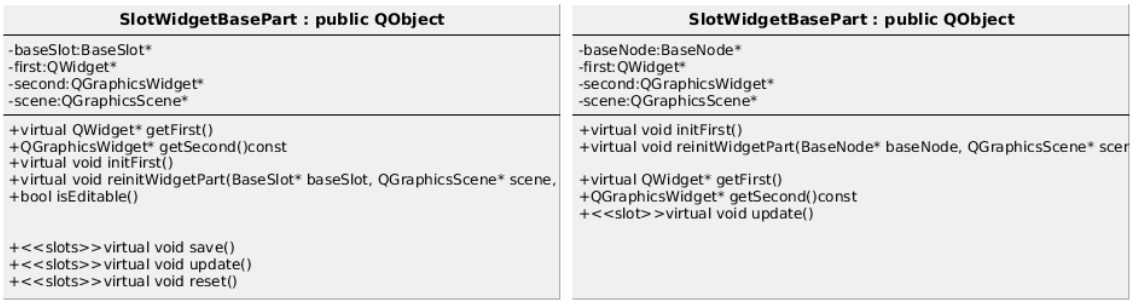
Ezen az ábrán egy Image típusú DataNodet láthatunk. Jelenleg a rendszer nem az InputList által átadott képet tartalmazza megában hanem a felhasználó által kiválasztotat. Továbbá látható, hogy a node isReady státusza true tehát azonnal végrehajtható (ez az alsó READY feliratból olvasható ki), és hogy az egy datab outputslotja is készen áll a végrehajtásra. Az input slotok mindig a node bal oldalán foglalnak helyet, még az output slotok a node jobb oldalán. Jelen esetben mint a 3 alapvető funkciógom elérhető, a clone, delete és update is.

4.4.6. NodeWidgetBasePart

Egy-egy alap NodeWidget elem megjelenítését és kezelését egy-egy NodeWidgetBasePart végzi. Osztálydiagramja lentebb látható. Két fontos adattagja van: first és second. Ahogy a diagrammon látható a first egy QWidget még a second egy QGraphicsWidget. NodeWidgetBasePart legfontosabb feladata, hogy eltároljon egy widgetet (ez lehet akár saját készítésű widget is). Majd az initFirst() függvénnyel inicializálja first nevű widgetet (töltse be a BaseNodeból a megfelelő adatokat), a művelet végén ezt a widgetet, pedig adja hozzá a scene pointer által mutatott QGraphicsScenehez. A GraphicsScenehez történő hozzáadás egy proxy osztályt ad vissza, amelynek memóriacímét letárolja a second pointerrel. A NodeWidgetBasePart a külvilág felé egy slotot biztosít: ez az update() slot, mellyel szinkronizálható a BaseNode és a first widget közötti adat tartalom. (A second widgetben automatikusan frissül a mutatott tartalom, hiszen az csak egy proxy osztály.) Példa NodeWidgetBasePart osztályokra:

Név	Feladat és eszköz
NodeWidgetTitle	Megjeleníti az aktuális BaseNode címét (Ez a cím a BaseNode nevéből kategóriájából és a ProcessViewen betöltött sorszámából adódik). (QLabel)
NodeWidgetPreviewPixmap	Előnézeti pixmap, DataNodek esetén a bemeneti kép jelenik meg itt, IndicatorNodek esetén az éppen kiszámolt kimeneti kép. (QPixmap és QLabel)
NodeWidgetStatusLab	A node isReady státusza jelenik meg itt. (QLabel)
NodeWidgetCloneButton	Egy nyomógomb aminek segítségével kérvényezheti a NodeWidget a ProcessChainView felé a klónozást. (QPushButton)
NodeWidgetDeleteButton	Egy nyomógomb aminek segítségével kérvényezheti a NodeWidget a ProcessChainView felé a saját maga eltávolítását. (QPushButton)
NodeWidgetBaseUpdateButton	Frissítés indul fut le a NodeWidgeten. (A felhasználó saját maga kezdeményezheti a NodeWidget manuális frissítését

1.5. táblázat: A NodeWidget részei



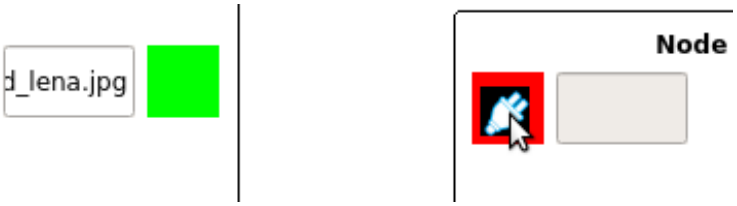
17. ábra: A NodeWidgetBasePart és a SlotWidgetBasePart osztályok diagramja

4.4.7. SlotWidgetBasePart

A SlotWidgetBasePart osztály működését tekíntve nagyon hasonló a NodeWidgetBaseParthoz. A különbség annyi, hogy ebben az esetben egy slot megjelenítéséről gondoskodunk, és abban az esetben ha az adott slot egy DataNode Outputslotja: akkor a slot által hordozott paramétert szerkeszthetővé tesszük. (Erre azért van szükség, hogy a ProcessChainbe más adat is kerülhessen ne csak az InputListről kapott kép.) Mivel új adattípusok is érkezhetnek a pluginekkel ezért a pluginekbe ilyen SlotWidgetBasePart osztályokat is csomagolhatunk. (Ahogy azt a Plugin rendszer bemutatása során leírtam.)

A SlotWidgetBasePart kezelése egyszerű: a bitosított editor felületen szerkeszthetjük az adatot: pl.: szám esetén ez egy számbeviteli mező lesz, ahova bevihetjük a kért számot. Amint az editor elveszíti a fókuszt a benne lévő adat mentésre kerül. Azonban ha biztosra szeretnénk menni jobb kattintással az editorban kiválaszthatjuk a kívánt funkciókat. 2 slotot úgy tudunk összekötni, hogy a kis slot állapotot jelző indikátor mezőt megfogjuk és fogd és vidd módszerrel egy másik slot indikátor mező fölött elengedjük. Ha jó slot összekötését kezdeményeztük, akkor a kapcsolat automatikusan létrejön a két node között.

Két slot közötti kapcsolat létrehozatalának pillanata. Fontos megjegyzés: Látható, hogy az output slot zöld tehát készen áll a végrehajtásra, még az inputslot piros tehát még nem érkezett rá adat.



18. ábra: Slotok összekapcsolása

5. A Fejlesztés részletei

5.1. Fejlesztési napló

A fejlesztés során iteratív fejlesztési módot követtem. Az alapvető jellemző metodikám az volt, hogy kiválaszottam egy modult, aminek a legfontosabb funkcióit vázlatosan megterveztem. Ha az adott modul első iterációjáról volt szó akkor egy külön teszt projektben készítettem el az első verziót. (Így lehetőségem, volt az új funkciókat úgy tesztelni esetlegesen újra tervezni átvariálni, hogy közben a rendszer többi részével nem kellett hosszas munkaórákat töltenem, hogy képes legyen együttműködni az esetlegesen később teljesen megváltoztatott architektúrával.) Az fejlesztés során összegyűlt tapasztalatokat bugokat, esetleges új funkció kéréseket mindig a következő iterációban vettem csak figyelembe. Egy iteráció hossza 1-3 nap volt. A következő felsorolásban vázolom, hogy melyik modulok, alrendszerek fejlesztése tartozott egy iterációba.

5.1.1. Iterációk

1. Node-Slot-Parameter rendszer kezdetei tervei és Proof of Concept implementációja. (template)
2. Input modul megtervezése és implementálása, Viewgui és settings libek elkészítése.
3. Node-Slot-Parameter rendszer teljes revíziója és újratervezése. (metatype), Qt Pluginrendszerének vizsgálata, és állatorvosi lovának elkészítése, a későbbi tervezés megkönnyítéséhez.
4. Process Chain tervezése és implementálása (körkereséssel és gráfelemzéssel), Node-Slot-Parameter rendszer tesztei és javítása, saját metatípussal rendelkező slotok tesztje.

5. OpenCV - Qt wrapper megtervezése, implementálása és tesztje
6. OpenCV - Qt wrapper revíziója
7. Pluginrendszer megtervezése és első plugin elkészítése
8. Teljes refaktor: Első mérföldkő működőképes Node-Slot-Parameter rendszer és Plugin rendszer, (OpenCV-Qt konverziók még mindig nem 100%osak)
9. Process Chain áttervezése és implemetálása
10. NodePalette elkészítése és NodeEditor első verziója
11. NodeEditor áttervezése és implementálása
12. NodePartWidgetek és SlotPartWidgetek megtervezése és implementálása
13. BaseScene és ProcessChainView szétválasztása
14. Második mérföldkő: kezdetleges GUI működik
15. GUI csiszolása, tesztek, bugfixek
16. Bugfix + alpha plugin fejlesztése
17. Deploy rendszerek előkészítése (dupla hosszúságú lett végül)
18. Bugfix, elmaradt apró funkciók pótlása
19. Deploy + Dokumentáció igazítása (papír alapú dokumentációk digitalizálása)

5.1.2. Névterek és libek

A legfontosabb modulokat, alrendszereket külön lib-be és névterekbe csomagoltam. A következő két táblázat ezeket mutatja be a legfontosabb szervező logikákkal együtt.
A namespaceket a 1.6 táblázat tartalmazza. A libeket a 1.7 táblázat tartalmazza.

5.1.3. Nehézségek

A projekt tervezése, implementálás során több nehézség is adódott. Az egyik jelentősebb a Node-Slot-Parameter rendszer kidolgozása körül volt a Slotok kapcsolatainak kezelése kapcsán. (Hogyan és mikor legyenek lebontva a kapcsolatok, hogy ne

Névtér	Leírás
IMP	Itt található minden, alapvető osztály, amelyek szorosan kapcsolódnak a Node-Slot-Parameter rendszerhez. Ebbe a névtérbe kerülnek a be a pluginekben fejlesztett Nodek is.
VIEWGUI	A legalapvetőbb megjelenítési funkciókat magukba záró osztályok lelőhelye.
BIMG	A BIMG rendszer fő névtere, itt található minden amit az IMP felett van architektúrálisan: Plugin és Input kezelés, ProcessChain, NodePalette stb.

1.6. táblázat: A BIMG rendszerben található névterek és szerveződésüknek legfontosabb logikája

Név	Funkció
bimg_imp	Gyakorlatilag a teljes Node-Slot-Paramter rendszer és a minden szükséges architekturális és GUI elem alapját képező osztály megtalálható itt, ami ahhoz szükséges egy saját plugin fejlesztéséhez. (statikus linkelésű)
guiview	Alapvető grafikus megjelenítők lelőhelye a QGraphicsScene és a QGraphicsView felett található osztályok, amelyek önállóan is hasznosíthatóak bármely GUI-s alkalmazáshoz. (statikus linkelésű)
settings_lib	QSettings felett működő minilib. (Jelenleg még inkább snippset.) (statikus linkelésű)
plugin_alpha	A BIMG magját képező nodek, feldolgozó egységet és a Qt-OpenCV konverziós megfeleltetések lelőhelye. (dinamikus linkelésű plugin)
qslog_lib	Egy nagy kompakt Qt-s loggerlib[28] amelyet razvanpetru publikált bsd licence alatt. (statikus linkelésű)

1.7. táblázat: A BIMG rendszert felépítő libek listája

fordulhasson elő inkonzisztens állapot. A másik probléma a tárol adatok konverziójával volt, hogy milyen módon lehetséges egy tetszőleges új saját típusra konvertáló függvényt regisztrálni a metatype rendszerbe. Mind a kettő problémát sikeresen megoldottam és a dolgozat korábbi fejezeteiben mindegyiket bemutattam.

A pluginrendszer sok gondolkodást és tervezést okozott, hogy hogyan lehet elegánsra és biztonságora megtervezni, hogy egy hibásan megírt node, vagy plugin ne okozzon túl nagy problémát. Erre csak részleges megoldás született: a rendszer minden kivételt elkap és megpróbálja feldolgozni (egy hibaüzenettel megállítja a feldolgozást és közli, hogy honnan érkezett a hiba). Azonban ez csak az hibaesetek egy részére hozott megoldást pl.: ha a nodeban szegmentálási hibatörténik akkor azt ilyen módon nem lehetséges jól kezelni. A megoldásra terveket már készítettem, azonban a jelenlegi architektúra jelentős szintű utólagos megváltoztatását igényelte

volna. Amire jelenelg sajnos nem volt lehetőségem. A terv lényege annyi, hogy a egy PluginContainer szolgáltatást hozunk létre a háttérben, amelyre a BIMG valamilyen módon felcsatlakozik, megadja a végre hajtani kívánt Node metatype-idjét és a bemeneti és kimeneti slotok adattípusát és jelenlegi állapotát, a service elvégzi a számításokat, majd visszatér az eredményekkel. A service mindig tárolja a legutolsó kérést ezért ha pl szegmentálási hibakeletkezik akkor képes az adott node tiltására. Amennyiben a pluginContainer adott időszakon belül nem küld választ a BIMG részére akkor a BIMG megállíthatja és újraindíthatja azt.

5.1.4. Tesztek

A fejlesztés során a különböző rendszerek, alrendszerek, modulok, osztályok és függvények folyamatosan tesztelve voltak. Ezek a tesztek jó része kézzel történt. Készült pár félautomata teszt eset is (ezek elsősorban a Node-Slot-Parameter rendszerhez készültek.

Alább látható egy teszteset kimenetele:

```
Starting /home/szendvics/szakdoli/src/bimg5/bimg/build-bimg_imp-↩
    Desktop_Qt_5_2_1_GCC_32bit_595617-Debug/bimg_imp...
##########################################
#Base Node system test
##########################################
DefaultData         1
DefaultIndicator    0
NewValue            11
TargetResult        22
##########################################
"#Init:                " PASSED!
"#Connectors:          " PASSED!
"#Setter&Conv:         " PASSED!
"#Getter&Conv:         " PASSED!
"#EXEC-DATA:           " PASSED!
"#EXEC-CONTROL:        " PASSED!
"#EXEC-VIEW:           " PASSED!
"#ISREADY FULL RESULT: " PASSED!
"#After Exec DataCheck:" PASSED!
GENERAL STATE           : true    9 / 9
```

```
##########################################
```

Ebben a tesztesetben a korai fejlesztésú Node-Slot-Parameter rendszert teszteltem. Konkrétan két slot közötti információáramlás helyességét azonos és különböző (de konvertálható adattípusok esetén). Egy teszt több kisebb tesztesetből áll. Először a teszt fejléce kerül megjelenítésre: a teszt neve és használt bemeneti paraméterek és a várt kimeneneti paraméterek. Utána következnek a tesztesetek eredményei, majd egy összegzés, hogy hányból hány teszt sikerült.

5.2. Az elkészült munka értékelése

Munkám során megterveztem és kifejlesztettem egy képfeldolgozást támogató keretrendszert, amely alkalmas képek egyedi vizsgálatára és kötegelt feldolgozására. A rendszer képes különböző (fájl)formátumokban és módokon vizualizálni az eredményképeket. A paraméterezhető képfeldolgozási algoritmusokat dinamikusan betöltődő modulok biztosítják, és egy egyszerű kompakt könnyen kezelhető grafikus felületen lehetőség van többlépcsős feldolgozás megvalósítására is. A rendszer ingyenes modern és nyíltforráskódú technológiák felhasználására épült.

A program még kűzd kisebb hibákkal, hiányosságokkal, gyermekbetegségekkel azonban az alap feladatai ellátásra alkalmas.

5.3. További fejlesztési lehetőségek

A következő pár pontban röviden bemutatom, hogy az adott modulok milyen további fejlesztési lehetőségeket foglalnak magukban.

5.3.1. Plugin rendszer

Egy bizonyos plugin szám felett elkerülhetet lesz a pluginek teljeskörű managelése. Ide értem, az aláírással és egyedi azonosítóval történő azonosításukat. Ennek legegyszerűbb oka az, hogy jól karban tartott pluginrendszernél is előfordulhatnak hibák, ilyenkor az adott plugint frissíteni kell. Erre célszerű egy automatikus frissítési modult elkészíteni. További ötletként felmerült egy központi plugin repository lehetősége is a jövőben, ahol a felhasználó szabadon összeválogathatja a számára szükséges plugineket.

5.3.2. GUI

A NodeWidgetek kompaktabbá tétele sokat segíthetne, a nagyobb feldolgozási láncok megalkotása során. (több nodet lehet látni egyszerűbb a komplexebb ProcessChainek kialakítása)

5.3.3. Optimalizálási lehetőségek

- A ProcessChain feldolgozás párhuzamosítása.
- A FileInput kezelő bélyegképes előnézeti funkciójának többszálúsítása.

5.4. Használati útmutató

Mivel korábban részletesen bemutattam a szoftver terveit, fejlesztését és működését, itt csak a legfontosabb hivatkozásokat gyűjtöm össze.

- **Input kezelés:** InputList
- **Node Palette használata:** NodePalette/NodePaletteForm
- **Node Editor használata:** ProcessChain, Scene&View - guiview lib, NodeWidget, NodeWidgetBasePart, SlotWidgetBasePart

5.5. Fejlesztői útmutató

A BIMG vagy valamely plugin fejlesztéséhez elsődlegesen ez a dolgozat szolgál információforrásul. A forráskódok Doxygenes kommenteléssel készültek ezért, abból generálható részletes dokumentáció. Egy előkészített DoxyGen konfigurációs fájl és egy legenerált teljes dokumentáció is megtalálható a mellékletben. A forráskódok könyvtárában: *Doxyfile* néven.

5.6. Telepítési útmutató

A BIMG esetében két féle telepítés mód járható.

5.6.1. Forráskódból

1. Ez esetben a program forráskódjára lesz szükség (amely a mellékletben megtalálható).

2. Telepített Qt 5.2.1es verzióra vagy újabbra.
3. Telepített 2.4.8as OpenCVre és forráskódjaira.
4. Nyissuk meg a bimg.pro fájlt QtCreatorban. Ellenőrizzük a subprojektekben a libeknek és includoknak az elérési útvonalát.
5. Fordítsuk le a szoftvert, és már használhatjuk is.

5.6.2. Binárisból

A mellékleten több bináris deploy is megtalálható összezippelve. Elnevezésük a következő képpen alakul: bimg_dátun_deploy_PLATFORM.zip Cél paltformok:

- Linux Mint Maya (Qt 5.2.1, OpenCV 2.4.8 32bit)
- Windows 7 (Qt 5.2.1, OpenCV 2.4.8 64bit)

Amint megtaláltuk a számunkra megfelelő zip fájlt tömörítsük ki egy testszőleges könyvtárba. Más dolgunk nincsen az archívumban megtalálható az összes szükséges függvénykönyvtár és konfigurációs fájl, amely a BIMG futtatásához szükséges.

6. Irodalomjegyzék

[1] Craig Larman (2004). *Applying UML and Patterns, Prentice Hall, 3 edition 6.7 (66)(127)*

[2] Jack J. Dongarra (1995). *Numerical Linear Algebra on High-Performance Computers (3)*

[3] http://www.valvesoftware.com/publications/2006/ SIGGRAPH06_Course_ShadingInValvesSourceEngine_Slides.pdf *Valve, Jason Mitchell (2007), Shading in Valve's Source Engine (37)*

[4] http://imagej.nih.gov/ij/ *ImageJ - Image Process and Analysis in Java*

[5] Tony J. Collinsm, *ImageJ for microscopy* BioTechniques 43:S25-S30 (July 2007)

[6] http://www.highmotionsoftware.com/products/imbatch
ImBatch - Batch Image Processing Software

[7] http://www.originlab.com/index.aspx?go=Products/Origin/DataAnalysis/ImageProcessing
OriginLab - Image Processing

[8] http://www.originlab.com/index.aspx?go=COMPANY/AboutUs
OriginLab - About Us

[9] http://www.originlab.hu/Originv9_USD_NEW_20121022_WEB.pdf
OriginLab - Licences

[10] http://www.ibm.com/developerworks/rational/library/4706.html
IBM - Capturing Architectural Requirements

[11] http://zone.ni.com/reference/en-XX/help/371361J-01/lvconcepts/blockdiagram/
NI - LabVIEW 2012 Help - Block Diagram

[12] https://docs.unrealengine.com/latest/INT/Engine/Blueprints/Editor/index.html
UDK4 - Blueprint Editor Reference

[13] http://qt.digia.com/About-Us/
QT - About

[14] http://qt-project.org/doc/
QT - Doc

[15] http://qt-project.org/forums
QT - Forums

[16] http://lists.qt-project.org/mailman/listinfo
QT - MailingLists

[17] http://qt.digia.com/Qt-in-Use/
QT Digia - In Use

[18] http://opencv.org/about.html
OpenCV - About

[19] http://blog.qt.digia.com/blog/2014/04/16/qt-5-2-over-1-million-downloads/
QT Digia - Over 1 million

[20] http://www.math.unipd.it/ tullio/IS-1/2007/Approfondimenti/SWEBOK.pdf
Guide to the Software Engineering Body of Knowledge, Chapter 2 - SOFTWARE R EQUIREMENTS

[21] http://www.ibm.com/developerworks/rational/library/4706.html#N10098
IBM - Capturing Architectural Requirements, Functional Requirements

[22] Ruth Malan and Dana Bredemeyer http://www.bredemeyer.com/pdf_files/NonFunctReq.PDF
Architecture Resources For Enterprise Advantage (2)

[23] http://opencv.org/about.html
OpenCV - About

[24] http://qt-project.org/doc/qt-5/qmetatype.html#Q_DECLARE_METATYPE
Qt - Q_DECLARE_METATYPE

[25] http://qt-project.org/doc/qt-5/qvariant.html#canConvert
Qt - QVariant konverziók

[26] http://qt-project.org/doc/qt-5/plugins-howto.html
Qt - Plugins

[27] http://www.ietf.org/rfc/rfc4122.txt
Standards Track - A UUID URN Namespace (RFC 4122)

[28] https://bitbucket.org/razvanpetru/qt-components/overview
QsLogger Lib

7. Mellékletek

7.1. Felhasználói útmutató

7.2. CD melléklet

A szakdolgozat CD mellékletének könyvtárszerkezete:

/VargaMarcell-DVLKHU-szakdolgozat.pdf

/szakdolgozat-forraskod

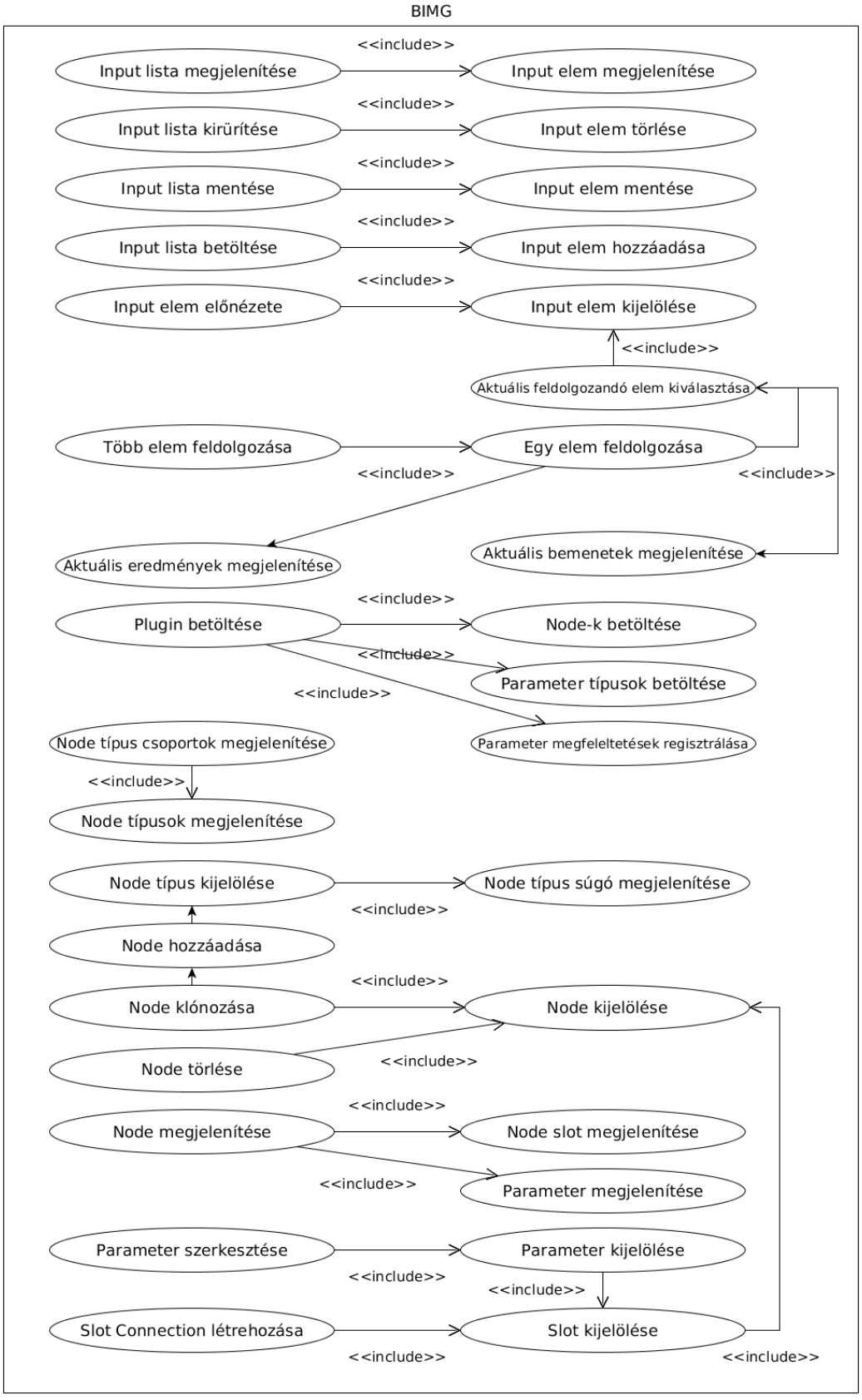
- /diagramok
- /szakdolgozat.tex

/rendszer-forraskod

- /src
  - /Szakdolgozat

/internetes-hivatkozasok

- /1_osszehasonlitas
- /2_kovetelmenyanalizis



19. ábra: A BIMG sematikus usecase diagramja